A Survey for Software Visualization

[1]So Young Moon, [2] R. Youngchul Kim, [3*]Chae Yun Seo

[1] *Department of CIC, Hongik University, Sejong, Korea,*
*{msy, bob,*chyun}@selab.hongik.ac.kr*

Abstract

It is an important issue of high-quality in software with frequently requirement changes on huge-scale codes and shorter time-to-market. Small sized IT companies still focus on code-based development across the domestic software industry, and also shift in development/testing processes and maturity measurements as a means of guaranteeing high-quality software. It remains the challenging issues of software quality about invisibility, increasing complexity and unfavorable development environment in small businesses. To address these problems, we introduce a code visualization technique that reduces the code complexity in source code. More mentioned about 1) recognizing bad development habits of his/her developer, and 2) improving software codes with visualization. This paper introduces how to visualize the inner structure of codes, and how to improve the quality of software codes.

***Keywords**: Code Complexity, Software Quality, Automatic Tool, SW Visualization, Reverse Engineering*

1. Introduction

Software has been widely used across diverse fields, but needs to guarantee the software quality of final products. However, its software invisibility and complexity as well as small and medium sized software development environment have depended on software quality [1]. Most of companies for high-quality software require 1) certifying the quality of software products. In domestic approaches, we have *Software Process* (SP) and Good Software (GS) certificates. In international approaches, Spice and CMMi are used to assure the quality of software. 2) Software development methodologies are used to develop high-quality software. 3) TMMi is used one of test processes for assessing test maturity level of test organization and quality enhancement. These approaches need to spend cost, and burden on small and medium sized IT companies. This paper shows high-quality software development to focus on code visualization based on reverse engineering [2]. In NIPA (National IT Industry Promotion Agency) software center, they also work on the software visualization technique with a view to: 1) identifying and modifying the problems of legacy codes; 2) providing a guideline for software developers' bad smells by a reverse engineering technique via code visualization; and 3) absenting developers or relevant documentation for legacy code maintenances. To visualize the internal code structure, this paper also introduces a software visualized tool by combining related open sources, i.e., Source Navigator, DOT Script and SQLite. This tool mechanism is applied to the software visualization to derive a visualized output from original code structures. Also, in complexity view, his/her programmer may perform refactoring to improve high-quality software.

This paper presents as follows. Chapter 2 mentions software visualization and reverse engineering in related studies. Chapter 3 describes the automatic tool mechanism for code visualization. Chapter 4 illustrates some examples. Chapter 5 mentions the conclusion.

---

* Corresponding Author
Received: Sep. 17, 2016, Revised: Sep. 22, 2016, Accepted: Sep. 26, 2016

2. RELATED WORK

Moon[11] mentioned this code visualization approach in her paper. We introduce her software visualization approach in this paper.

A. *SW Visualization:* Most companies tried to develop, test, and certify their software product for guaranteeing high-quality software. These, however, are challenging to venture startups, small-medium sized IT companies, 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 of IT venture startups, small-medium sized IT companies, and even established companies by a lack of personnel and financial resources [3]. Software visualization may be one technique chosen for the better software quality control and maintenance by visualizing source codes.

First, visualization can mention the most challenging aspect of software development, i.e., invisibility, putting an entire software development process into perspective for controlling the quality of both software and its development process. Second, SW visualization aims to manage source codes and development processes, specifically involving visualization. An entire process of software development needs to be efficiently managed to produce good software. SW visualization enables clarification of goals in line with guidelines, system-based efficient development activities, and continuous monitoring and controlling via visualization. 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 reduce development cost and to ultimately attain corporate competitiveness [1].

Second, providing options for documentation of diverse outputs piled up inside the system in the course of development, SW visualization minimizes related workloads, whilst maximizing the usability.

B. *Reverse Engineering:*

Forward engineering starts with outputs from requirements specification, and progressively goes through analysis and design to implement software products. On the contrary, reverse engineering is the process of analyzing source code with the objective of recovering its design and specification. Reverse engineering can extract design information from source code [4]. The objective of reverse engineering is to derive the design or specification of a system from its source code. Reverse engineering 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].

Using this technique, this paper introduces the visualization to analyze and understand software without helping of developers, and thus to inspect legacy systems. This reverse engineering is used for software visualization to describe traceability and complexity issues, to restore lost information, to detect adverse effects, and consequently to reuse software. This paper introduces the open-source based automatic tool mechanism for visualization and reverse engineering in chapter 3 in detail.

3. The Automatic Tool Mechanism

Figure 1 shows the tool-chain process built with open sources.

A. *Open source*

The automatic tool mechanism process is based on open sources described below.

Source Navigator [6]: Source Navigator NG is a source code analysis tool. With it, we can edit our source code, display relationships between classes and functions and members, and display call trees. We can navigate our source code and easily get to declarations or implementations of functions, variables and macros (commonly called "symbols"). This helps discover and map unknown source codes for enhancement or maintenance tasks. SQLite [7]: SQLite is an in-process library that implements a self-contained, server-less, zero-configuration, transactional SQL database engine. The

code for SQLite is in the public domain and is thus free for use for any purpose, commercial or private. SQLite is an embedded SQL database engine. Unlike most other SQL databases, SQLite does not have a separate server process. SQLite reads and writes directly to ordinary disk files. A complete SQL database with multiple tables, indices, triggers, and views is contained in a single disk file. Graphviz [8]: Graphviz is open source graph visualization software. Graph visualization is a way of representing structural information as diagrams of abstract graphs and networks. It consists of five internal programs such as dot, neato, fdp, sfdp, twopi, circo. We use the dot. Dot is "hierarchical" or layered drawings of directed graphs. This is a default tool to use if edges have directionality.

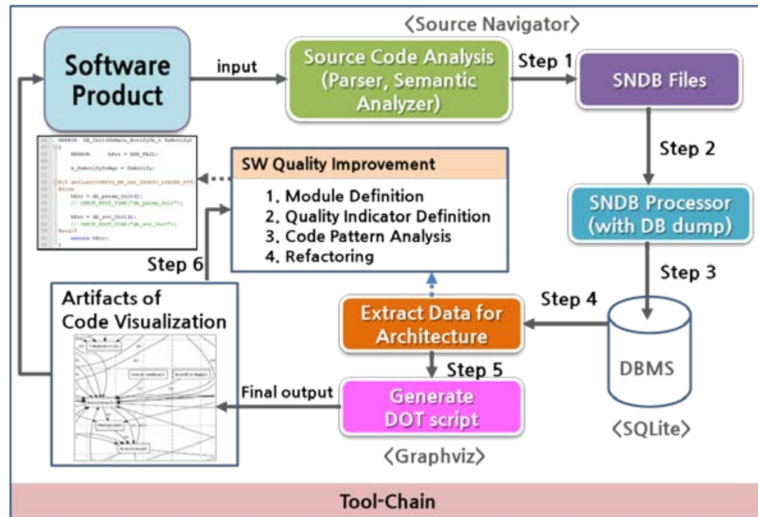


Figure 1. Automatic tool mechanism[11]

B. Process of Automatic tool mechanism

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).

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. Table 1 outlines the source code information contained in different types of SNDB files generated in SN.

Table 1. SHOIN syntax and semantics

Type	*C/C++ and Java*	*Type*	*C/C++ and Java*
Cl	Class, Struct	fu	Function
Con	#define	gv	Global variable
	const	iv	Instance variable
	static final	lv	Local variable
E	Enum	ma	Macro
Ec	Enum value	fr	Friend
Fd	Function declaration		

Step 3: Create DB from SNDB Processor

The data analyzed in Step 2 are 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 ex-tracts 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.

1) 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.

2) 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]. 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, and 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. These are expiations of coupling and cohesion.

< Coupling>

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., FunctionCode, Switch, Tag, 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

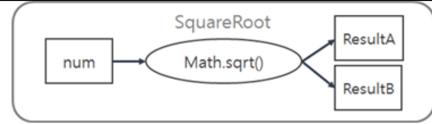
In content coupling, a module directly refers to or modifies the internal functions or data of another module.

<Cohesion>

a) Functional Cohesion

Every item in a module is related to a single task. Functional is the strongest cohesion.

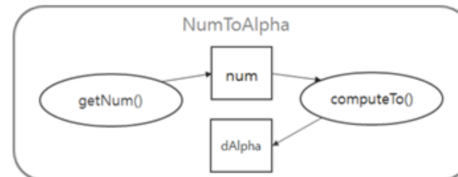
```
public class ComputeSquare{
  public static void main(String[] args) {
    int num = 16;
    double ResultA, ResultB;
    ResultA = Math.sqrt(num);
    ResultB = -Math.sqrt(num);
    ...
  }
```



b) Sequential Cohesion

Sequential cohesion is when operations in a module must occur in a pre-specified order and each operation depends on results generated from preceding modules.

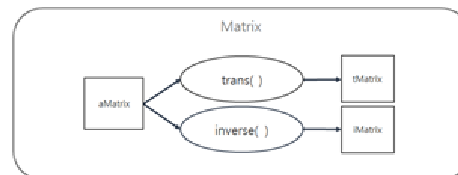
```
public class NumToAlpha {
  public int getNum(){
    ...
    return num;
  }
  public String computeTo(int _num){
      ...
  }
  public void displayAlpha(String _dAlpha){
    System.out.println(_dAlpha);
  }
  public static void main(String[] args) {
    int num;
    String dAlpha;
    ...
  }
}
```



c) Communicational Cohesion

All Elements work on the same piece of data.

```
public class Matrix {
  void ComputeMatrix(int tMatrix[][], int
vMatrix[][]){
    for(int i=0;i<5;i++){
    ...
    }
    tMatrix = trans(aMatrix);

    iMatrix = inverse(aMatrix);
  }
}
```



d) Procedural Cohesion

All related items must be performed in a certain order.

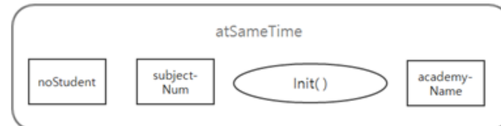
```
public class sendEmail {
  public void writeCent() {
  ...
  }
  public void writeGreet() {
  ...
  }
  public void read() {
  ...
  }
```

```
  public static void main(String[] args) throws
Exception {
    Object omail = new Object();
    omail.writeCent();
    omail.writeGreet();
    omail.read();
  }
}
```

e) Temporal Cohesion

Items are grouped in a module because the items need to occur at nearly the same time.

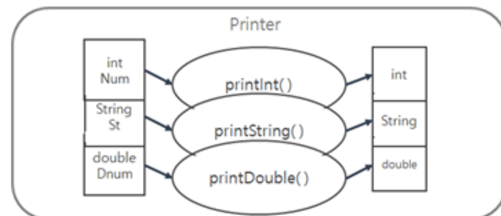
```
public class atSameTime {
  int noStudent;
  int subjectNum;
  String academyName;

  void Init(){
    noStudent = 0;
    subjectNum = 0;
    academyName = "Hongik";
  }
}
```



f) Logical

Items are grouped in a module because they do the same kinds of things.

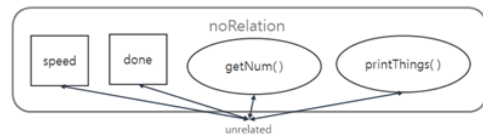
```
public class Printer {
  void printInt(int _Num){
    System.out.println(_Num);
  }
  void printString(String _St){
    System.out.println(_St);
  }
  void printDouble(double _Dnum){
    System.out.println(_Dnum);
  }
  public static void main(String[] args) {
    int Num;
    String St;
    Double Dnum;     ...
  }
}
```



g) Coincidental Cohesion

Items are in a module simply because they happen to fall together. There is no relationship.

```
public class noRelation {
  void getNum(){
    int num;
    Scanner sc = new Scanner(System.in);
    num = sc.nextInt();
  }
  void printThings(){
    System.out.println("Name");
  }
  public static void main(String[] args) {
    int speed;
    int score = 26;
    Coincidental cr = new Coincidental();

    cr.getNum();
    score = speed*0.1;
```



```
        cr.printThings();
    }
}
```

3) 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 inter-class relationships.

4) Refactoring

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

4. CASE STUDY

The coupling and cohesion between classes with this approach are visualized with DOT. Figure 2 shows stamp coupling graph in code visualization. The class identifiers and their cohesion are marked inside the graph nodes. The arrows connecting nodes indicate the coupling of the classes pointed to by the starting class. In Figure 2, the coupling of the class SystemAction with the class DMethodCall is 48.

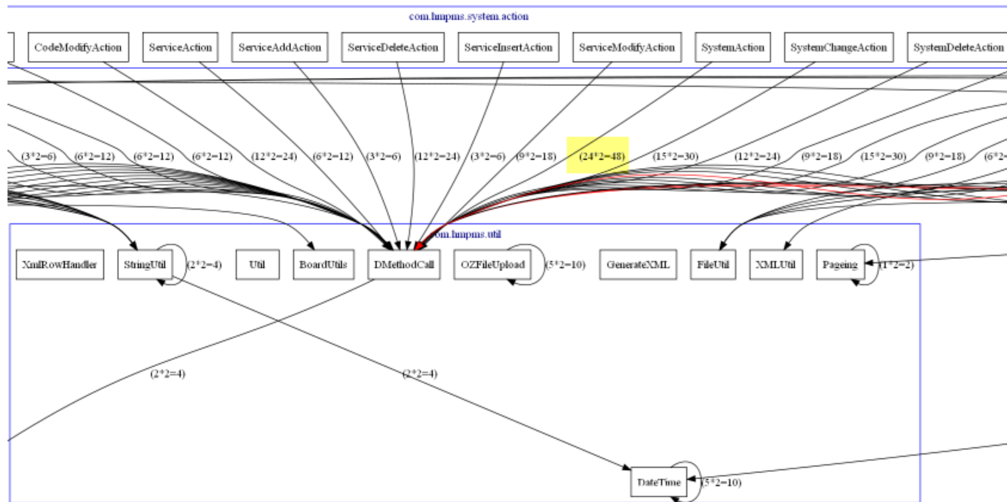


Figure 2. Stamp coupling graph in Code Visualization

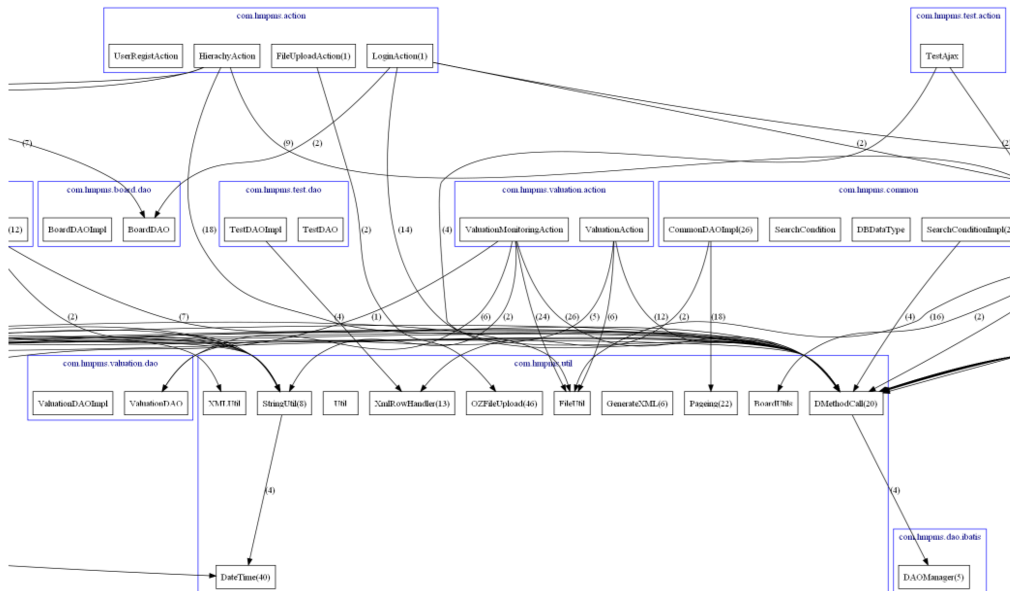


Figure 3. Code Visualization

Based on the cohesion and coupling of classes visualized with this approach, Figure 3 shows the relation of call between modules. Also, they are represented as the units of a package.

5. CONCLUSION AND FUTURE WORK

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, small-medium sized companies in software development. For the purpose of high-quality software, this approach focuses on visualizing problems of existing codes and recognizing software developers' bad smells. As highlighting development/testing and maturity measurement as a means of delivering high-quality software leads developers to additional workloads other than development, it cannot be an alternative for enhancing the quality of legacy systems. To address this issue, this automatic tool mechanism can visualize the complexity of codes. This approach enables even developers of bad smells to lessen the code complexity with refactoring.

6. Acknowledgements

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).

7. References

[1] NIPA SW Engineering Center, "SW Development Quality Management Manual (SW Visualization)", pp. 3-4, 2013.
[2] Steve McConnell, Code Complete (2nd ed.), Microsoft Press, 2001.
[3] Geon-Hee Kang, R. Young Chul Kim, Geun Sang Yi, Young Soo Kim, Yong B. Park, Hyun Seung Son, "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, February 2015.
[4] Roger S, Pressman, Software Engineering: a practitioner's Approach (7th ed.), McGrawHill, 2010.
[5] Ivan Sommerville, Software Engineering (6th ed.), Pearson Education, 2001.
[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, "A Study on Tool-Chain for statically analyzing Object Oriented Code", KCC2014, pp.463-465, 2014.
[10] Dustin Boswell, Trevor Foucher, The Art of Readable Code, O'Reilly, 2011.
[11] So Young Moon, R. Youngchul Kim," Code Structure Visualization with A Tool-Chain Method" Research India Publications(SCOPUS) Vol. 10, No. 90 214-218 15.12