객체지향 역공학을 위한 소프트웨어 복잡도 측정 기법

(A Software Complexity Measurement Technique for Object-Oriented Reverse Engineering)

김 종 완 † 황 종 선 ††
(Jongwan Kim) (Chong-Sun Hwang)

요 약 지난 10여년간 객체지향 코드의 관리 및 분석을 위해 객체지향 소프트웨어 시스템에 대한 다양한 복잡도 계산 기법들이 제안되었다. 이러한 기법들은 WMC(Weighted Methods per Class), LCOM(Lack of Cohesion in Methods)과 같이 소스코드 분석을 기반으로 한다. 기존 기법들의 한계는 코드에서 함수의 개수만 계산한다는 것이다. 본 논문에서는 함수의 파라메타 개수, 반환값 여부 그리고 자료형까지도 확인하는 새로운 가중치 기법을 제안하며, 이를 역공학에 적용한다. 또한 역공학과정에서 객체지향 코드를 위한 클래스 복잡도 계산 지침을 제공하기위해 인터페이스에 가중치를 부여하는 효율적인 복잡도 측정 기법을 제안한다. 제안기법인 ECC(Enhanced Class Complexity)는 C++환경에서 일관성 있고 정확한 결과를 보여준다.

키워드 : 소프트웨어 메트릭스, 클래스 복잡도 메트릭스, 객체지향 역공학

Abstract Over the last decade, numerous complexity measurement techniques for Object-Oriented (OO) software system have been proposed for managing the effects of OO codes. These techniques may be based on source code analysis such as WMC (Weighted Methods per Class) and LCOM (Lack of Cohesion in Methods). The techniques are limited to count the number of functions (C++). However, we suggested a new weighted method that checks the number of parameters, the return value and its data type. Then we addressed an effective complexity measurement technique based on the weight of class interfaces to provide guidelines for measuring the class complexity of OO codes in reverse engineering. The results of this research show that the proposed complexity measurement technique ECC(Enhanced Class Complexity) is consistent and accurate in C++ environment.

Key words : Software metrics, class complexity metrics, object-oriented reverse engineering

1. Introduction

Reverse engineering makes programs and constructs associative deliverables for maintenance, refactoring, reuse and documentation of legacy codes. Recent reverse engineering techniques can be conducted by analyzing program structure. The results are determined lexically, syntactically and semantically [1]. Most of systems do not have sufficient historical data and documents for management and engineering. Therefore, reverse engineering must start from analyzing source codes, execution files or documents of a system without its blueprint. In this paper, we concentrate on the source codes of a system.

In reverse engineering, first, the analyst must understand the system. Then, the analyst must determine how to measure the complexity of function-level and class-level granularity comprised of C++ member functions and class definitions respectively. The member functions of C++ are functions. We identify the functionality of a function in the system by checking intersections between functions and classes. Functions and

† 학생회원 : 고려대학교 컴퓨터학과
wany@disys.korea.ac.kr
†† 종신회원 : 고려대학교 컴퓨터학과 교수
hwang@disys.korea.ac.kr
논문접수 : 2005년 1월 26일
심사완료 : 2005년 8월 3일

classes are distinguished by major and minor functionalities.

To process reverse engineering, we check the flow of codes without seeing detailed logics of class methods. We look up the function calls in the codes to capture the major part in the system. This is a Gray Box Testing(GBT). In spite of GBT is interface-based scheme such as BBT GBT uses knowledge of implementation [2].

White Box Testing(WBT) can access private state and make use of implementation knowledge. More effort is required with object-oriented methodology in the analysis stage compared with traditional methodologies for class complexity measurement. Software measurement is fundamental in the field of reverse engineering. Traditional metrics, which are Halstead's software science and McCabe's cyclomatic complexity, have been used in the procedural paradigm [3]. Object-Oriented metrics have been proposed in the object-oriented world but are not sufficient to characterize the manner of OO systems.

Chidamber & Kemerer(C & K) metrics are widely used currently [4]. C & K metrics have some deficiencies in reverse engineering. Although some deficiencies have been evaluated using criteria in [3], it is not sufficient to analyze for reverse engineering.

We cannot find easly the paper that was applied OO metrics to reverse engineering so far. In this paper, we employed the GBT method from [2] and modified the result of [3]. We suggested a new complexity measurement method to enhance the metrics for reverse engineering. We tested a new method with C++ codes and and compared with C & K WMC. The contributions of this research are as follows.

- Analyze OO systems using GBT
- Employ existing OO metrics for reverse engineering.
- Propose ECC as a new method that makes more weighted results than C & K WMC. As a result, analysts can identify the major and minor parts of a system.

The remainder of this paper is organized as follows. In section 2, we discuss literature reviews about system analysis and C & K metrics. In section 3, we present the mapping process of minor and major parts, and then describe the new proposed ECC method. Verification of the technique is described in section 4. Lastly, section 5 summarizes this research.

2. Literature Review

This section reviews code mapping to analyze subsystems of a system and Chidamber & Kemerer metrics adapted for reverse engineering. Because reverse engineering analyzes source codes, deliverables or manuals, this paper does not consider dynamic execution, which is called by dynamic binding such as virtual functions in C++ but is focused on each component of a system that represents the static complexity of the system. Static complexity measurement means that it does not consider run-time execution and considers complexity in the source code level only. [2] describes three categories of static complexity models - regression analysis models, discriminant analysis models and hybrid metric models. The following is brief description of the models.

- Regression analysis models: With given metric values of a system, calculate the number of potential faults in the software system such as lines of code, debugging time and fault repair effort based on the metric values.
- Discriminant analysis models: With given metric values of a system, classify the system, which has a few changes or potentially many changes by probability.
- Hybrid metric models: Generate a single complexity metric value that can be used as an indication of system complexity using raw metric values of a system such as the micro/macro complexity metric MMC and the Maintainability Index (MI).

Among the three types of models, hybrid metric models allocate resources in the software testing [5,6]. Therefore, it is convenient to use hybrid metric models because they do not require data on historical defects. Variables, methods and objects define complexity and it is useful in Chidamber &

Kemerer metrics. Nevertheless, the proposed metrics use different methods to get weighted values.

The features of Chidamber & Kemerer metrics are various such as WMC(Weighted Methods per Class), LCOM(Lack of Cohesion in Methods) [4,7,8].

WMC is the number of methods of a class for static complexity measurement. LCOM is the number of method pairs.

$$\text{Let } P = \{(I_i, I_j) \mid I_i \cap I_j = 0\} \quad (1)$$

$$Q = \{(I_i, I_j) \mid I_i \cap I_j <> 0\} \quad (2)$$

$$LCOM = |P| - |Q|, \text{ if } |P| > |Q| = 0, \text{ otherwise} \quad (3)$$

Let $\{I_j\}$ be a set of instance variables used by method M_i. There are n sets $\{I_1\}$, ···, $\{I_n\}$ of instances [4]. LCOM is defined as (1), (2) and (3).

3. Enhanced Class Complexity Measurement Technique

We must decide on the first base line how to measure complexity of C++ function level in the complexity measurement. Complexity of the class level must be decided from C++ class definitions.

To identify subsystems from a system, we perform the GBT on object-oriented codes, which had developed in C++. In GBT or white box testing each of encryption/decryption algorithms can be forced to execute and we can get the code portions of the subsystem for the major and minor functionalities. An OO subsystem has some difficulties in mapping between the major and minor parts.

In this section, we will show the result of mapping between functions, classes and minor parts by MAP3 and MAP4, but MAP1 and MAP2 do not appear here. Next, Fig. 1 shows relations between minor parts, classes and functions in a subsystem. Relative complexity and GBT method are employed from [2] to analyze which classes are associated each other in the system. The items analyzed compose a new method for complexity measurement.

Let MR be a set of major parts *mr*, and NR a set of minor parts *nr* that is joined to the major parts. Sub is a set of subsystems *s* in NR. In subsystem *s*, we identify minor parts $nr_i \in$ NR (*i* is a number to identify each part).

Let MAP3 be a relation of F × NR, in which F is a function in a class. Table 1 shows an example of mapping functions to minor parts. '√' indicates 'mapped' and '-' does 'not mapped.'

Table 1 Mapping functions to minor parts (MAP3)

F × NR	*nr1*	*nr2*	*nr3*	*nr4*
F1	√	-	√	-
F2	-	√	√	√
F3	-	√	-	√
F4	√	-	√	√

Let MAP4 be a relation of C × NR, in which C is a class in a subsystem. Table 2 shows an example of mapping classes to minor parts.

Table 2 Mapping classes to minor parts (MAP4)

C × NR	*nr1*	*nr2*	*nr3*	*nr4*
C1	√	√	-	-
C2	√	-	√	√
C3	-	√	-	√
C4	√	√	√	√

Therefore, minor part *nr1* is set {C1, C2, C4, F1, F4}, *nr2*={C1, C3, C4, F2, F3}, *nr3*={C2, C4, F1, F2, F4} and *nr4*={C2, C3, C4, F2, F3, F4}. As major parts, we identified *mr1* and *mr2* through the analysis step, which are {*nr1, nr3, nr4*} and {*nr2, nr4*} respectively. Therefore, major part *mr1* is the union of *nr1*={F1, F4}, *nr3*={F1, F2, F4} and *nr4*={F2, F3, F4} in MAP3. In the same manner, the class mapping sets are *nr1*={C1, C2, C4}, *nr3*={C2, C4} and *nr4*={C2, C3, C4}, so *mr1* is {C1, C2, C3, C4, F1, F2, F3, F4}. Accordingly, major part *mr2* is set {C1, C2, C3, C4, F2, F3, F4}. From these sets, we can check the relationship between minor parts, functions and classes in Fig 1. C4 is common between *nr1, nr2, nr3* and *nr4*, but because there is not enough space to put C4 we separate it in each part.

In this paper, we propose the ECC(Enhanced Class Complexity) metrics to measure class complexity but ECC_1 is not full ECC in this step. It

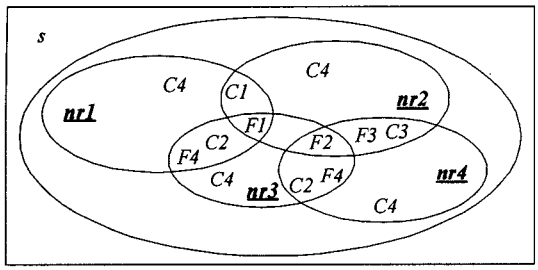

Fig. 1 The relationship between minor parts, functions and classes

will be changed to ECC later. WMC defined by Chidamber & Kemerer is comprised of the sum of weighted methods per class without measurement of internal complexity of functions and variables. That is, WMC has a simple summation of functions in a class. In addition, we suggest another factor *cv* for ECC. *cv* is comprised of the number of class variables.

$$ECC_1 = WMC + cv \quad (4)$$

Class complexity according to ECC_1 is the sum of the numbers of instance variables. That is, it is the sum of all local methods and all variables in a class. ECC_1 calculates the number of parameters in a function and data types as a measurement of function complexity. The return value and its data type are also part of the base line. We consider call-by-value and call-by-reference styles in a function call. The data types of C++ are composed primitives, pointers and user defined types. Since both array and struct are user-defined types, a set of primitives and pointers, the weighted value is assigned as one or three for primitive or pointers respectively using direct rating method (DRM) [9].

DRM assigns a large value to the most important item and compares it with other items. The weighted ratios are 1/4 or 3/4 according to the importance of primitive or pointer type. In case of array and struct type, the weighted value is calculated by summing up the internal data types.

Since it is hard to understand of multiple pointers, we assign a value increased by 3 to a pointer of a pointer variable cumulatively. (e.g. in case of int** ptr, the weighted value is 6). We surveyed to get weighted values by distributing questionnaires to developers and reverse engineering researchers who has many experiences.

Thus, ECC1 is comprised as follows. *fv* is the sum of function parameters and the return value. If a function has five parameters and a return value as primitive, it will have *fv* = 6. If all of them are pointers, *fv* = 18.

$$ECC_1 = WMC + cv + fv \quad (5)$$

Based on Table 1 and 2, let us assume that class C1 is mapped into function F1={nr1, nr3}, class C2 into function F2={nr2, nr3, nr4} and C1={nr1, nr2}. That is, F1∈C1. In the next expression, *cf* is the number of the union set between class *nr* and function *nr*.

$$ECC = WMC + cv + fv + cf \quad (6)$$

The numbers of classes and functions express frequencies of use in a subsystem. Thus, analyzers can select and analyze preliminarily the minor part for reverse engineering.

4. Verifications of the Technique

A system was analyzed by the ECC method described above and an analysis target was selected preliminarily. The example codes show only one class. First, we found three minor parts and they are as in Table 3 below.

The result of mapping classes to minor parts is in Table 4 and Table 5 show the complexity using ECC.

Table 3 Mapping functions to minor parts

F × N	nr1	nr2	nr3	Function	Class
MountDirectory	-	-	√	F1	C1
UnmountDirectory	-	-	√	F2	
ShowAddCFSUserDialog	√	-	-	F3	
ShowAddCFSGroupDialog	√	-	-	F4	
Refresh	-	√	√	F5	
Clear	-	√	√	F6	
Socket	√	-	-	F1	C2
Bind	-	-	√	F2	

Table 4 Mapping classes to minor parts

C × N	nr1	nr2	nr3	Class
CCrbCFSDirListCtrl	√	√	√	C1
CasyncSocket	√	-	√	C2

Table 5 Examination of a subsystem

Class	C & K WMC	ECC
CCrbCFSDirListCtrl	6	30
CasyncSocket	2	22
CsyncObj	1	9
CCriticalSection	1	10
CsingleLock	2	16

According to formulas presented above, the results of set mapping are F1={nr1, nr3}, F2={nr3}, F3={nr1}, F4={nr1}, F5={nr2, nr3}, F6={nr2, nr3} and C1={nr1, nr2, nr3}. We can calculate ECC straightforward. cf is 8 because (function∪class) =1+1+1+1+2+2.

$$ECC=WMC+cv+fv+cf=6+12+4+8=30$$

The virtual functions of C++ are not included in static complexity measurement because they are called dynamically in run-time. The parameters in a function have weighted values in this measurement. For a pointer, the weight is increased by 3. If the primitive data type factor is not a pointer, 1 is added.

The return value is processed in the same way. Then the number of union with functions, classes and minor parts is added to the ECC. The 'CCrbCFSDirListCtrl' class is computed by the following source code:

```
01: class CCrbCFSDirListCtrl : public CUIListCtrl
02: {
03: private:
04:     CCrbAPClient        *m_pCrbAPClient;
05:     CCrbUserResource    *m_pUserResource;
06:     CCrbGroupResource   *m_pGroupResource;
07:     CCrbCFSResource     *m_pCFSResource;
08: protected:
09:     virtual void OnCreateHandler();
10:     virtual void OnDestroyHandler();
11:     virtual void OnItemChangedHandler(NM_LISTVIEW
          * pNMListView);
12: public:
13:     void MountDirectory(int nItem);
14:     void UnmountDirectory(int nItem);
15:     void ShowAddCFSUserDialog(int nItem);
16:     void ShowAddCFSGroupDialog(int nItem);
17:     void Refresh();
18:     void Clear();
19: };
```

5. Implications and Conclusions

In this paper, we measured the class complexity using a new weighted method that checks the number of parameters, the return value and its data type in order to avoid the simplicity of old methods that check only the number of functions. Moreover, to establish guidelines for class complexity measurement for C++ source codes (Object-Oriented) in reverse engineering, we suggested a measurement technique. This paper showed practical results and adjusted a new method for complexity measurement by putting weighted values on class interface.

We tested source codes to prove some strong points over C & K method using improved measurement method ECC. In addition, the complexity measurement was used as a method to verify the complexity and system dependency of analysis target sources in reverse engineering. We can select more important functions or classes using ECC. In reverse engineering, it takes several hours to check the detailed contents of target sources because the analysts did not develop the sources. Therefore, we used GBT method to check functions and classes in the system.

During GBT, we could not see the detailed logics and check the cyclometic complexity of functions. In this case, WMC will be equivalent to the number of local methods. Therefore, if we use the ECC measurement method, the complexity of classes can be checked more thoroughly and used in decision making for reverse engineering.

Further study is necessary to calculate comparative complexity between each class and function. We will get more accurate and reliable results from system analysis in comparative complexity measurement.

References

[1] Jean-Marc DeBaud, Bijith Moopen, Spencer Rugaber, Domain Analysis and Reverse Engineering, pp. 326-335, IEEE, 1994.

[2] Jianqiang Zhuo, Paul Oman, Ramkumar Pichai, Sujay Sahni, Using Relative Complexity to Allocate Resources in Gray-Box Testing of Object-Oriented Code, pp. 74-81, IEEE METRICS, 1997.

[3] N. V. Balasubramanian, Object-oriented Metrics, pp. 30-34, in Proc. Int. Asia-Pacific Conf. Software Engineering, 1996.

[4] Chidamber S. R., Kemerer C. F., Towards a Metrics Suit for Object Oriented Design, pp. 197-211, OOPSLA, 1991.

[5] Warren Harrison, Using Software Metrics to Allocate Testing Resources, pp. 93-105, Journal of Management Information Systems, Vol. 4, No. 4, 1988.

[6] John C. Munson and Taghi M. Khoshgoftaar, Applications of Relative Complexity Metric for Software Project Management, pp. 283-291, J. Syst. Software, Vol. 12, No. 3, 1990.

[7] Chidamber S. R., Kemerer C. F., A Metric Suit for Object Oriented Design, pp. 476-493, IEEE Transactions on Software Engineering, 20(6), 1994.

[8] Tom Mens, Michele Lanza, A graph-based metamodel for object-oriented software metrics, Electronic Notes in Theoretical Computer Science 72 No.2, 2002.

[9] D. von Winterfeldt and W. Edwards. Decision Analysis and Behavioral Research, Cambridge University Press, Cambridge, 1986.

Jongwan Kim

Jongwan Kim is a Ph.D. candidate at the Department of Computer Science and Engineering, Korea University. He works for Distributed Systems Laboratory. He has more than 10 years field experiences as a developer and technician for COM, COM+. He has developed various applications and associated with numerous projects including Medical Information System using PDA, All Mighty IT Co. Ltd., Configuration Management System Dev. For Software Reuse Artifact Management Based on Reverse Engineering, Korea Science and Engineering Foundation (KOSEF). His research interests include Mobile Data Management, Location Based Services, Object- Oriented technologies, Component-Based Development.

Chong-Sun Hwang

Chong-Sun Hwang received the M.S. degree in Mathematics from Korea University, Korea in 1970, and Ph. D. degree in Statics and Computer Science from University of Georgia I 1978. From 1978 to 1980, he was an Associate Professor at South Carolina Lander State University. He is currently a Full Professor in the Department of Computer Science and Engineering at Korea University, Seoul, Korea. Since 1995, he has been a dean in the Graduate School of Computer Science and Technology at Korea University. His research interests include distributed systems, distributed algorithms, and mobile computing systems.