KIPS Tr. Software and Data Eng.
Vol.6, No.5 pp.229~234 pISSN: 2287-5905

https://doi.org/10.3745/KTSDE.2017.6.5.229

Plug-in Diverse Parsers Within Code Visualization System with Redefining the Coupling and Cohesion in the Object-Oriented Paradigm

Lee Jin Hyub[†] · Park Ji Hun[†] · Byun Eun Young[†] · Son Hyun Seung[††] · Seo Chae Yun[†††] · R. Young Chul Kim[††††]

ABSTRACT

Because of the invisible nature of software and the bad coding habits (bad smell) of the existing developers, there are many redundant codes and unnecessary codes, which increases the complexity and makes it difficult to upgrade software. Therefore, it is required a code visualization so that developers can easily and automatically identify the complexity of the source code. To do this, it is necessary to construct SW visualization tool based on open source software and redefine the coupling and cohesion according to the object oriented viewpoint. Specially to identify a bad smell code pattern, we suggest how to plug-in diverse parsers within our tool. In this paper, through redefining coupling and cohesion from an object oriented perspective, we will extract bad smell code patterns within source code from inputting any pattern into the tool.

Keywords : Complexity, Coupling, Cohesion, Plug-in

객체지향 관점의 결합도 & 응집도 재정의와 코드 가시화 시스템내 파서 플러그인화 구현

이 진 협[†] · 박 지 훈[†] · 변 은 영[†] · 손 현 승[††] · 서 채 연[†††] · 김 영 철[††††]

요 약

소프트웨어는 비가시적 특성과 기존 개발자들의 나쁜 코딩 습관인 중복된 코드, 불필요한 코드 등이 많아 복잡도가 높아져 소프트웨어의 고품질화가 저해된다. 그러므로 개발자가 소스코드의 복잡도를 쉽게 자동 식별하는 코드 가시화가 필요하다. 이를 위해, 공개 소스 기반의 가시화 도구를 구축하였다. 특히 나쁜 코드 패턴들을 식별하기 위해, 다양한 파서를 plug-in하는 방법을 제안한다. 또한 객체지향 관점에 맞는 결합도와 응집도 재정의를 통해, 자유로운 패턴을 입력하여 개발자가 원하는 나쁜 패턴을 추출하고자 한다.

키워드 : 복잡도, 결합도, 응집도, 플러그인

1. 서 론

SW 산업이 발전함에 따라 SW의 크기도 점차 커지고 있지만, 소프트웨어의 비가시성 때문에 특정 인원이나 조직이 개발 과정 전반을 모두 파악하기 어렵다. SW 전반에 걸친 가시성 확보를 위한 SW 가시화가 필요하다. SW 가시화는 SW 개발 과정에 대한 SW 엔지니어뿐만 아니라 SW 개발 경험이 없더라도 프로젝트 진척상황과 품질수준 파악이 쉬워진다[1, 2].

본 논문에서는 객체지향(JAVA) 언어의 소스코드 내 결합도와 응집도를 객체지향에 맞게 재 정의하고 예시코드를 통해 그래프로 가시화한다. 기존의 오픈소스인 소스네비게이터를 활용한 방법 외에 다른 파서를 이용하여 소스코드를 분석하여 활용할 수 있도록 구축한 시스템을 설명한다. 이를 통해 개발자가 원하는 파서를 사용하거나 여러 파서를 이용하여 가시화한 결과물에 대한 신뢰성을 높이고자 한다.

본 논문의 구성은 다음과 같다. 2장에서는 기존 툴체인과 파서의 종류, 코드 복잡도에 대해 설명하고, 3장에서는 객체지향관점에서 결합도와 응집도를 재 정의한다. 그리고 4장에서 다양한 파서(SourceNavigator와 XCodeParser)를 적용하여 구축한 파서 플러그인 툴체인 시스템에 대해 설명한다. 그리고 5장에서는 시스템을 활용한 결합도와 응집도 가시화를 소개하고, 마지막 6장에서 결론 순으로 언급한다.

※ 이 논문은 2015년 교육부와 한국연구재단의 지역혁신창의인력양성사업의 지원을 받아 수행된 연구임(NRF-2015H1C1A1035548).
※ 이 논문은 2016년도 한국정보처리학회 추계학술발표대회에서 '기존 절차식 파라다임의 응집도 개념을 객체 내부 코드 응집도 비교 및 가시화 구현'의 제목으로 발표된 논문을 확장한 것임.
† 준 회 원 : 홍익대학교 소프트웨어전공 석사과정
†† 정 회 원 : 홍익대학교 메카트로닉스 연구센터 박사후연구원
††† 비 회 원 : 선문대학교 IT교육학부 강의전담계약교수
†††† 정 회 원 : 홍익대학교 컴퓨터정보통신공학과 교수
Manuscript Received : January 24, 2017
Accepted : January 26, 2017
* Corresponding Author : R. Young Chul Kim(bob@hongik.ac.kr)

2. 관련 연구

2.1 기존 툴체인

기존 툴체인에서는 절차식 언어에서의 응집도와 객체지향식 언어에서의 응집도를 예시코드와 소스코드의 내부 흐름도로 가시화하여 비교하였다[3]. 기존 툴체인에서는 응집도 가시화를 위해 아래와 같이 4개의 단계로 구성되었다.

Code Input 단계에서는 타겟 소스코드를 오픈소스 코드 분석도구인 소스네비게이터에 입력한다. 이때, 타겟 소스코드는 객체지향 언어인 자바 기반 코드이다.

Analysis, save DB 단계에서는 입력된 소스코드를 소스네비게이터로 분석한다. 분석 결과로는 *.cl, *.by *.lv 등과 같은 바이너리 파일이 생성된다. 이 파일들을 소스네비게이터의 DBdump로 정보를 추출하여 데이터베이스에 저장한다.

Analysis Structure 단계에서는 데이터베이스에 저장된 정보를 툴체인의 정의에 따라 재해석하여 *.gv 파일을 생성한다. 이 파일을 GraphViz에 입력한다.

마지막으로, Visualization 단계에서는 이전 단계에서 재해석한 *.gv 파일을 GraphViz를 이용해 그래프로 가시화한다.

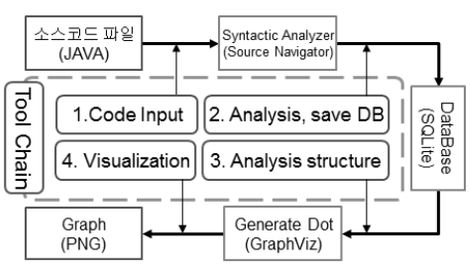

Fig. 1. Existing Toolchain System Structure Map

```
try {
① br = new BufferedReader(new FileReader(inFile));
  String line;
  String[] arr;

  while ((line = br.readLine()) != null) {

      String method_name="";
      DB db = new DB();
      Connection conn = db.getConnection();
      Statement stat = conn.createStatement();
      Statement stat2 = conn.createStatement();
② ResultSet rs = stat.executeQuery("select * from SNDB_BY");
  while(rs.next()){
       String referred_symbol_name = rs.getString("referred_symbol_name");
       ResultSet rs2 = stat2.executeQuery("select method_name from SNDB_MD w
       while(rs2.next()){
           method_name = rs2.getString("method_name");
           if(line.contains(method_name)){
               if(!total.contains(method_name)){
                   total += method_name+" ";}
           }
③ if(line.contains("return")){
       arr = line.split(" ");
       st = arr[1];
```

Fig. 2. Additional Steps to Extract "return"

기본적인 가시화를 위한 툴체인은 Fig. 1과 같이 수행되는 것이 올바른 순서이다. 하지만 기존 툴체인은 필요한 데이터 몇 가지를 찾을 수 없는 문제가 있다. 클래스 내부의 흐름을 나타낸 부분이 있는데 예를 들어, 변수를 메소드에 입력하고 return된 값을 다른 메소드나 제어문에 입력하고 return 받는 형태이다. 이 툴체인에서 사용한 파서인 소스네비게이터가 추출해 주는 분석데이터로는 필요한 return 값을 추출할 수 없었다.

그래서 Fig. 2 처럼 전체 소스코드 파일들을 다시 읽어(①), 소스코드 내에 return을 찾아내어(③), 소스네비게이터로부터 분석되어 저장된 데이터베이스 중 관련 필드에 추가해주는 방법(②)으로 수행하였다.

2.2 다양한 파서의 종류

소스코드 분석을 위한 파서는 종류가 다양하다. 여러 종류의 소스코드들을 분석하여 다양한 형태로 추출해 준다. 파서 프로그램들의 예로는 오픈소스인 SourceNavigator, JavaParser, 자체제작 도구인 XCodeParser 등이 있다. Table 1은 파서의 종류와 특징이다.

Table 1. Parser Types And Features

Type	Feature
Source Navigator	Open source code analysis tool. It can analyze various kinds of code such as C, C ++, and Java. Enter code and generate binary file as result, database with DBdump.
XCodeParser	C, C ++, and Java code are analyzed by the self-produced tool of Hongik University software engineering laboratory and made into ASTM (Abstract Syntax Tree Metamodel) file.
JavaParser	Open source code analysis tool. Target only the source code coded in Java. It is based on JDT ASTParser, and it is made in Abstract Syntax Tree (AST) form.

2.3 코드 복잡도

코드 복잡도란 소프트웨어 프로그램의 복잡도를 측정하는 체계이다. 일반적으로 복잡도 메트릭을 통해 측정한 결과 수치가 높은 경우, 많은 에러가 발생할 가능성이 높아진다. 복잡도를 측정하는 방법으로는 LOC(Line of Code), 맥케이브 복잡도 메트릭(McCabe's complexity metrics), 할스테드 메트릭(Halstead's metrics), 결합도(Coupling), 응집도(Cohesion) 등이 있다[4]. 이들을 간단히 설명하면 다음 Table 2와 같다.

Table 2. Complexity Metric Types and Simple Descriptions

Type	Description
Line of Code	It measures the number of lines of source code. While performing the same function, the code of a shorter number of lines can be considered to have a low complexity.
McCabe's Complexity Metrics	The number of disconnected parts of the graph, the number of edges, and the number of nodes are substituted into the cyclomatic complexity formula to quantify. The higher the quantity, the higher the code complexity.
Halstead's Metrics	It shows the code complexity by measuring program length, lexical number, volume, difficulty, implementation effort index, implementation time, number of estimated bugs.
Coupling	It is a measure of the degree of data transfer and reference between modules, indicating the degree to which one module relies on another module. There are six types of coupling, data, stamp, control, external, common, and content coupling.
Cohesion	It shows how a module of a program performs only one function, and measures the degree to which a component in a module is aggregated. There are seven types of cohesion: functional, sequential, exchangeable, procedural, temporal, logical, and coincidental [5].

3. 객체지향 기반 결합도 & 응집도 재정의

2001년 Daniel Rodriguez[6]에 의하면 기존의 절차식 언어기반 소프트웨어는 캡슐화, 상속, 다형성 등의 개념들이 없다. 그러나 객체지향 개념은 캡슐화, 상속, 다형성 등의 개념들을 포함하고 있다. 그 예로 객체, 클래스 컴포넌트, 패키지 등이 있다. 이처럼 결합도와 응집도는 기존의 절차식 언어에서의 Function 관점과 객체지향에서의 객체 관점은 차이가 있다. 그러므로 객체지향 언어의 특성에 맞는 재정의가 필요하다.

결합도는 객체지향 언어에서 클래스 또는 객체를 기본 단위로 하며, 서로 간의 데이터 전달, 파일 공유 등을 나타낸다. 때문에 절차식 언어(C)에는 없는 클래스/객체 단위의 개념이 객체지향 언어(Java)에 있으므로 재정의 될 필요가 있다. Table 3은 결합도의 기존 정의와 객체지향 관점에 맞게 재 정의한 표이다.

Table 3. Redefining Coupling in Object-oriented Paradigm

Type	Existing Definition	Redefinition
Data Coupling	There are no or one parameters to exchange between modules,	There are no parameters to exchange between classes / objects,
Stamp Coupling	It exchanges complex data structures such as arrays between modules and modules as parameters.	The parameters to be passed between classes / objects are arrays or objects.
Control Coupling	Depending on the parameters that are exchanged between the modules, they affect the module.	The parameter passed between the class / object is used as the condition of the condition statement.
External Coupling	Communication is made between modules using external files or files.	External files and data between classes / objects are transferred and used.
Common Coupling	Global variables are shared between modules.	The parameters to be passed between classes / objects are declared as static.
Content Coupling	One module modifies or relies on the internal behavior of another module.	Changes the internal properties of other objects, such as getter / setter between classes / objects.

결합도 중에서 자료 결합도가 가장 낮은 결합력을 가지며, 내용 결합도가 가장 높은 결합력을 갖는다. 결합력이 낮아질수록 모듈 간 주고받는 데이터 개수가 적어지거나, 데이터 타입이 간단해지며, 각 기능이 독립적이게 되어 코드의 품질이 높아진다.

응집도는 객체지향 언어에서 클래스 또는 객체 단위 내에서 속성과 메소드 또는 기타 연산들의 관계를 나타낸다. 따라서 응집도 또한 재정의 될 필요가 있다. Table 4는 응집도의 기존 정의와 객체지향 관점에 맞게 재 정의한 표이다.

낮은 응집도를 가진 모듈은 하나의 모듈 내에서 여러 기능을 한다는 의미이며, 반대로 높은 응집도를 가진 모듈일수록 한 가지 기능만 수행 한다. 즉, 높은 응집도를 가질수록 품질 또한 좋다고 볼 수 있다.

Table 4. Redefining Cohesion in Object-oriented Paradigm

Type	Existing Definition	Redefinition
Functional Cohesion	All components in the module perform only one function.	All variables and components within an object perform a single task.
Sequential Cohesion	The result of one execution in the module is used as the input value of another execution.	The methods in the object are executed sequentially, and each method depends on the previous method.
Communication Cohesion	Perform other functions using the same input / output in the module.	The components in the object act on some of the same data.
Procedural Cohesion	The components in the module do not share input and output, but perform them in order.	The configuration methods within an object are interrelated and are performed in a specific order.
Temporal Cohesion	Components that are not associated with the module perform operations at a certain point in time.	Components within an object perform different functions together at the same time.
Logical Cohesion	The components in the module have similar characteristics or are classified into specific types.	The components within the object perform a logically related task or similar function.
Coincidental Cohesion	Each component in the module performs a task that is completely unrelated to each other.	All functions and components within an object perform functions that are not related to each other.

4. Plug-in Diverse Parser based on Toolchain

본 장에서는 관련 연구에서 언급한 문제점들을 보완하고 더 나아가 사용자가 파서를 선택하여 원하는 데이터를 가시화 할 수 있도록 XCodeParser를 추가하였다. 개선된 Toolchain 시스템에 대한 구성도는 Fig. 3과 같다.

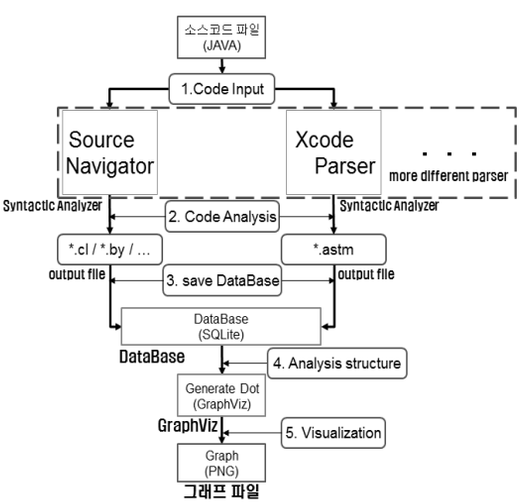

Fig. 3. Improved Toolchain System Structure Map

Fig. 3에서 향상된 시스템은 기존의 Toolchain 시스템보다 조금 더 세분화해서 5단계로 이루어져 있다. 파서 부분의 큰

점선으로 된 네모는 SourceNavigator와 XCodeParser 이 외에 다른 파서를 plug-in하여 활용할 수 있다는 의미이다.

4.1 툴 체인 절차

Input Code 단계에서는 툴체인 내의 getData에서 타겟소스 코드를 입력 받는다. 프로그램 실행 전 사용자가 선택한 파서(SourceNavigator 또는 XCodeParser)로 입력받은 데이터를 전달한다. 이때, 타겟 소스코드는 객체지향 언어인 자바 기반 코드이다.

Code Analysis 단계에서는 입력된 소스코드를 Source Navigator로 분석하고, 그 결과 *.cl, *.by *.lv 등과 같은 바이너리 파일들이 생성된다. 그리고 XCodeParser로 분석한 결과로는 입력한 소스코드 파일의 개수만큼 *.astm 파일이 생성되는데, 파일의 이름은 “입력된 각 소스코드 파일의 이름.astm”이 된다[7].

Save DB 단계에서 바이너리 파일들은 소스네비게이터의 DBDump로 정보를 추출하여 데이터베이스에 저장되고, *.astm 파일들은 따로 정보를 추출해서 데이터베이스에 저장해야 한다.

Analysis Structure 단계에서는 기존 시스템과 동일하게 데이터베이스에 저장된 정보를 Toolchain의 정의에 따라 재해석하여 *.gv 파일을 생성한다. 이 파일을 GraphViz에 입력한다.

마지막으로, Visualization 단계에서는 기존 시스템처럼 이전 단계에서 재해석한 *.gv 파일을 GraphViz를 이용해 그래프로 가시화한다.

4.2 Plug-In을 위한 DB 추출 쿼리

파서 플러그인화의 어려움은 파서마다 분석하여 DB에 저장하는 방법이 다르다는 것이다. 예를 들면, SourceNavigator 로부터 생성된 데이터베이스와 XCodeParser로부터 생성된 데이터베이스의 column 값들이 같은 종류의 데이터라도 전혀 다른 명칭으로 정의되어 있다. 이는 Fig. 4와 Fig. 5를 통해 더 자세히 알 수 있다.

1) SourceNavigator

```
String query = "select refer_class_name,referred_class_name,referred_type"
        + "referred_symbol_name,refer_argument_types "
        + "from SNDB_BY where refer_type='mi' and referred_type='mi'"
        + "or referred_type='iv' "
        + "order by referred_class_name asc, refer_class_name asc";
```

Fig. 4. Query Statement to Extract Information from DB Generated from Binary File

Fig. 4는 바이너리 파일에서 추출한 정보를 입력한 데이터베이스에서 가시화를 위한 정보를 추출하는 쿼리문으로, call 관계와 파라미터 타입, 파라미터 이름을 가져오는 부분이다.

2) XCodeParser

Fig. 5는 XCodeParser로부터 생성된 ASTM파일에서 추출한 정보를 입력한 데이터베이스에서 가시화를 위한 정보를 추출하는 쿼리문이다. 이 문장 또한 call 관계 추출을 위해 클래스 멤버인 메소드와 클래스, 호출자 등의 정보를 가져온다.

```
String module_query = "select ASTM_link.mem_name as CALLER,"
        + "ASTM_link.call_name as CALLEE , "
        + "ASTM_contents.class_name, ASTM_contents.par_type, "
        + "ASTM_link.file_name as CALLER_file,"
        + "ASTM_contents.file_name as CALLEE_file, "
        + "count(*) "
        + "from ASTM_link,ASTM_contents "
        + "where ASTM_link.call_name = ASTM_contents.mem_name "
        + "group by ASTM_link.mem_name, ASTM_link.call_name ";
```

Fig. 5. Query Statement to Extract Information from DB Generated from ASTM File

Fig. 6. Improved Toolchain System Launch Screen

Fig. 6은 개선된 Toolchain 시스템의 실행화면이다. 사용 방법은 먼저, 타겟소스 경로에는 분석할 소스코드의 경로를 입력한다. 이때, *.java 파일 또는 이클립스 프로젝트의 src 같은 소스코드 최상위 폴더 두가지 다 입력 가능하다. 출력파일경로는 중간 결과물인 *.cl / *.by 등 과 *.astm 파일들, 그리고 가시화를 위한 소스코드 *.gv 파일, 마지막으로 최종 결과물인 가시화 그래프를 저장할 경로를 입력한다. 미입력 시에는 자동으로 C드라이브에 폴더를 생성하여 입력된다. 프로젝트 이름은 결과물인 가시화 그래프 생성시 파일 이름을 입력한다.

어떤 파서를 사용할 것인지 선택하면, 하단에 해당하는 파서의 라디오버튼들이 활성화된다. 활성화된 버튼들은 클래스 다이어그램, 결합도와 응집도, 결합도만, 응집도만, 총 4개가 있다. 이들 중 하나를 선택하면 실행 버튼이 활성화되고, 실행 버튼을 클릭하여 그래프를 생성한다.

5. 코드 복잡도 가시화

기존에 소스네비게이터로 추출한 데이터만으로 객체지향 개념을 가시화하기에는 한계가 있다는 것을 언급했다. 그래서 Plug-in 형식으로 추가한 XCodeParser로 재 정의하다 결합도와 응집도를 추출하였다.

Equation (1)은 결합도와 응집도를 산출한 계산식이다. 개발자가 결합 정도와 응집 정도를 쉽게 파악할 수 있도록 수치화 한 것이다. 결합도의 총합은 클래스 간에 해당하는 결합 종류를 모두 합산한 것이다. 응집도의 총합은 클래스 내 메소드 간에 해당하는 응집 종류를 모두 합산 한 것이다. 가시화를 위해 사용된 각 결합도들과 각 응집도들의 가중치는 아직 정확한 수치가 정해져 있지 않기 때문에 임의로 정하였다[8].

$$\sum_{i=0}^{n} co = co_d + co_s + co_c + co_e + co_m + co_n \quad \text{... 결합도의 총합}$$

$$\sum_{i=0}^{m} ce = ce_f + ce_s + ce_m + ce_p + ce_t + ce_l + ce_c \quad \text{... 응집도의 총합} \qquad (1)$$

Fig. 7 ~ Fig. 9는 가시화에 Equation (1) 공식을 적용한 것이다.

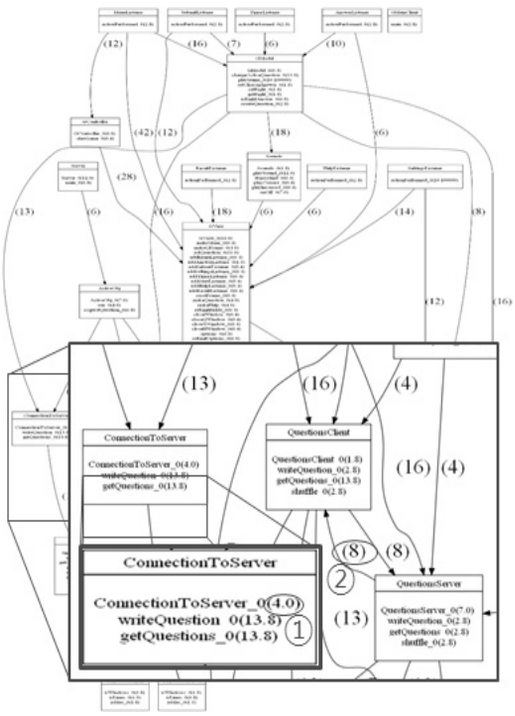

Fig. 7. Coupling and Cohesion Visualization Graph

Fig. 7은 재 정의하다 결합도와 응집도를 하나의 그림으로 가시화한 것이다.

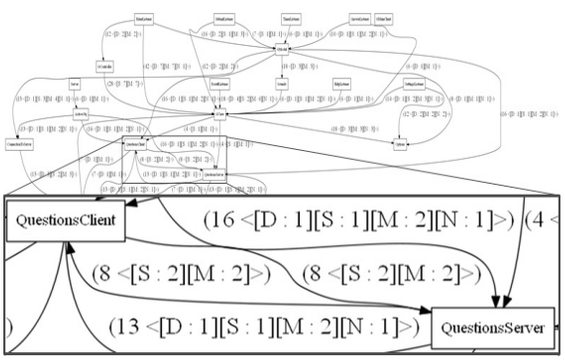

Fig. 8. Coupling Only Visualization Graph

클래스와 메소드간의 포함관계는 클래스 다이어그램 형태로 나타내었다. ① 표시는 해당 메소드의 내부 응집도들의 가중치 값들을 모두 합한 수치, ② 표시는 각 클래스 간에 결합도들의 가중치 값들을 모두 합한 수치이다.

Fig. 8은 결합도만 가시화한 것이다. 타원형으로 표시해둔 괄호 부분 내에서 8은 총 결합도 수치이고, 꺾음 괄호(<, >) 사이에서 S는 스탬프 결합도 2개, M은 공통결합도가 2개를 의미한다. 이외의 결합도 관계를 나타낸 다른 괄호 내에는 데이터 결합도 등 여러 개가 나와 있는 것을 보면 정상적으로 데이터가 추출됨을 알 수 있다.

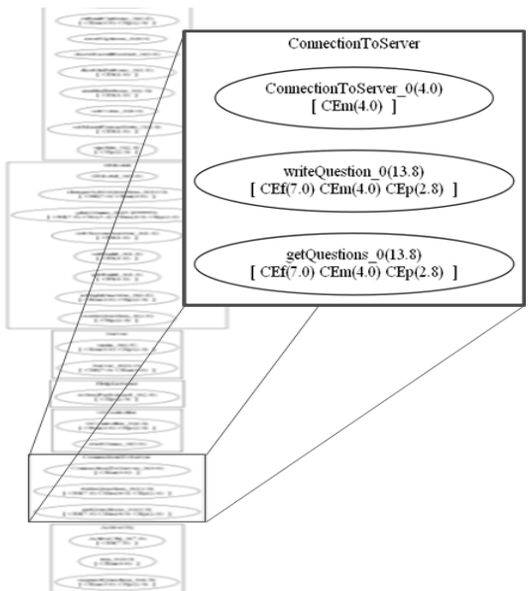

Fig. 9. Cohesion Only Visualization Graph

Fig. 9는 응집도만 가시화한 것이다. 결합도와는 다르게 클래스 간에 화살표가 없어 전체 그래프가 세로 혹은 가로로 길게 생성된다. 전체 그래프 중에서 Fig. 7에서 확대했던 ConnectionToServer 클래스만 확대 하였다. 클래스 내부의 메소드들이 있고 메소드 안에는 메소드 이름과 메소드 내 전체 응집도가 합산된 수치, 대괄호 내에는 어떤 응집도 종류가 해당되는지와 그 수치가 있다.

6. 결 론

소프트웨어의 크기가 계속적으로 커짐에 따라 소스코드의 복잡도 또한 더욱 복잡해지고 있다. 코드의 복잡도를 측정하는 지표들에는 순환복잡도와 결합도, 응집도 등이 있다. 본 논문은 기존 결합도와 응집도의 개념을 객체지향 관점에 맞게 재정의 하고, 예시코드를 가시화하였다.

또한 기존 연구에서의 문제점을 보완하고 개발자가 원하는 파서를 선택하여 가시화 할 수 있도록 시스템을 개선하였다. 이는 차후에 다른 파서 또한 Plug-in 형식으로 덧붙일 수 있다. 따라서 현재 시스템에서는 SourceNavigator와 XCodeParser로만 이루어져 있지만 앞으로 이클립스에서 기본으로 제공하는 ASTParser도 분석하여 추가할 예정이다. 또한 4장에서 언급한 바와 같이 파서마다 각각 생성하는 데이터베이스가 완전 다르다. 앞으로는 이를 공통된 부분, 다른 부분을 분석하여 통일된 하나의 데이터베이스 형태로 개선할 예정이다.

References

[1] NIPA SW Engineering Center, “SW Development Quality Management Manual(SW Visualization),” 2013. 12.

[2] Haeun Kwon, Hyun Seung Son, Chae Yun Seo, Youngsoo Kim, Byung Ho Park, and R. Youngchul Kim, “A study on Comparing Object Oriented Paradigm with the Cohesion and Coupling mechanism between Traditional modules,” *Korean Institute of Information Scientists and Engineers*, Vol.21 No.2, pp.556-559, 2014. 06.

[3] Telecommunications Technology Association Dictionary of Information and Communication Terms [Internet], http://terms.tta.or.kr/ dictionary/dictionaryView.do?kor_subject=%EB%B3%B5%EC%9E%A1%EB%8F%84.

[4] Jin-Hyub Lee, Chae-Yun Seo, Hyun-Seung Son, and R. Young Chul Kim, “Visual Implementation & Comparison of Internal Object Code with cohesion concept of the traditional procedural paradigm,” *Korea Information Processing Society*, Vol.23 No2, pp.487-489, 2016. 11.

[5] J. Eder, G. Kappel, and M. Schrefl, “Coupling and Cohesion in Object-Oriented Systems,” Technical Report, Univ. of Klagenfurt, 1994.

[6] D. Rodriguez and R. Harrison, “An overview of object-oriented design metrics,” *Proc. of the Conference on Software Technology and Engineering Practice (STEP)*, 2001.

[7] Eun-Young Byun, S.Y. Moon, C.Y. Seo, R. Young Chul Kim, and Hyun-Seung Son, “Comparison between xCodeParser and Open source tool for Software Visualization,” *ICT Platform Society*, Vol.4, pp.34-38, 2016. 07.

[8] Eun-Young Byun, Bo-Kyung Park, Woo-Sung Jang, and R. Young-Chul Kim, “Constructing Software Modernization System based on Open source software for a valuable module identification,” Korean Institute of Information Scientists and Engineers, pp.404-406, 2016. 12.

이 진 협

e-mail : ljh@selab.hongik.ac.kr
2010년～2016년 홍익대학교
컴퓨터정보통신공학과(학사)
2016년～현 재 홍익대학교
소프트웨어전공 석사과정
관심분야 : 소프트웨어 가시화 및 자동화 도구 개발, 오픈소스 기반 플러그인

박 지 훈

e-mail : pjh@selab.hongik.ac.kr
2010년～2016년 홍익대학교
컴퓨터정보통신공학과(학사)
2017년～현 재 홍익대학교
소프트웨어전공 석사과정
관심분야 : 역공학, 유지보수 우선순위 및 도구 개발

변 은 영

e-mail : eybyun@selab.hongik.ac.kr
2011년～2016년 홍익대학교
컴퓨터정보통신공학과(학사)
2016년～현 재 홍익대학교
소프트웨어전공 석사과정
관심분야 : 소프트웨어 재사용성 및 가시화

손 현 승

e-mail : son@selab.hongik.ac.kr
1999년～2015년 홍익대학교
소프트웨어공학(학·석·박사)
2015년～현 재 홍익대학교 메카트로닉스 연구센터 박사후연구원
관심분야 : 임베디드 소프트웨어 자동화 도구 개발, 소프트웨어 프로세스 및 가시화, 메타모델 설계 및 모델 변환, 모델 검증 기법 연구, 한국형 테스트 성숙도 모델, 시큐어 코딩

서 채 연

e-mail : chyun@selab.hongik.ac.kr
2003년～2014년 홍익대학교
소프트웨어공학(석·박사)
2007년～2013년 ㈜맥스컴 연구원
2016년～현 재 선문대학교 IT교육학부 강의전담계약교수
관심분야 : 비즈니스 프로세스 모델링, 의료 비즈니스 데이터베이스 스키마 자동화, 메타 모델

김 영 철

e-mail : bob@hongik.ac.kr
2000년 LG산전 중앙연구소 Embedded System 부장
2001년～현 재 홍익대학교 컴퓨터정보통신공학 교수
관심분야 : 테스트 성숙도 모델(TMM), 임베디드 소프트웨어 개발 방법론, 모델 기반 테스팅, 메타모델, 비즈니스 프로세스 모델, 사용자 행위 분석 방법론, 신재생에너지 통합관리 시스템