기존 절차식 파라다임의 응집도 개념을 객체 내부 코드 응집도 비교 및 가시화 구현

이진협[1*], 서채연[2*], 손현승[3*], 김영철[4*]
*홍익대학교 소프트웨어공학 연구실
e-mail: { [1]ljh, [2]chyun, [3]son, [4]bob }@selab.hongik.ac.kr

Visual Implementation & Comparison of Internal Object Code with cohesion concept of the traditional procedural paradigm

Jin-Hyub Lee*, Chae-Yun Seo*, Hyun-Seung Son*, R. Young Chul Kim*
*Software Engineering Laboratory, Hong-ik University

요 약

기존의 소프트웨어 개발자는 객체지향 내에서 나쁜 코드 습관으로 코드 자체의 결합도와 응집도를 고려하지 못 한다. 또한 SW 비가시성으로 인해, SW 내 복잡도 및 품질관리 등이 어렵다. 본 논문에서는 SW 복잡도 및 재사용 향상을 위해, 기존 절차식 모듈 관점 보다는 객체지향 메카니즘으로 응집도의 개념과 이를 통한 내부 코드 응집도 비교 및 가시화 구현하였다. 이는 내부 객체 코드의 응집도를 통해, 객체 내부 및 객체간의 복잡도 인식으로 재사용성과 코드 문제점 확보에 활용하고자 한다.

1. 서론

SW 품질 관리는 SW의 비가시적인 특성 때문에 어렵다. SW 개발 과정에서 발생하는 문제점은 개발 후반에 문제가 발생 할수록 이를 해결하기 위한 비용과 노력이 증가한다. 그래서 개발 과정 전반에 걸친 가시성 확보를 위한 노력이 요구된다. 2001년 Daniel Rodriguez[1]에 의하면 기존의 절차식 언어 기반 소프트웨어는 캡슐화, 상속, 다형성 등의 개념들이 없어, 객체지향 언어의 특성에 맞는 품질 관리 방안이 필요하다[1,2]. 그러므로 기존의 절차식 파라다임의 응집도를 객체지향 메카니즘으로 적용해야 한다. 또한 이를 시각적으로 표현하여 쉽게 이해하도록 가시화하면 개발과정에서 발생한 코드 복잡도를 줄일 수 있다[3]. 가시화를 위한 품질 지표에는 복잡도, 결합도, 응집도가 있다.

본 논문에서는 절차식(C) 언어와 객체지향(JAVA) 언어의 소스코드 내 응집도를 그래프를 통해 가시화하고, 비교한다. 2장은 응집도에 대해 설명한다. 3장은 절차식, 객체지향에서의 응집도 예시코드와 그래프를 비교한다. 4장은 결론 및 향후연구를 언급한다.

2. 관련연구

응집도란 모듈 구성 요소들이 얼마나 연관되었는지를 측정하는 척도이다. 높은 응집도를 가진 모듈은 하나의 기본적인 기능만을 수행하는 여러 개로 나누어지지 않는 모듈을 의미한다[4]. 응집도에는 기능적, 순차적, 교환적, 절차적, 시간적, 논리적, 우연적, 7가지의 응집도가 있다[5]. 각 응집도에 대한 자세한 설명은 표1과 같다.

<표 1> 응집도 종류와 정의

응집도	정의
기능적 응집도	모듈 내부의 모든 기능과 구성요소들이 한 기능을 수행
순차적 응집도	모듈 내 한 구성요소의 출력이 다른 구성 요소의 입력
교환적 응집도	모듈이 여러 가지 기능을 수행하며, 모듈 내 구성 요소들이 같은 입력 자료를 이용하거나 출력 데이터가 동일
절차적 응집도	모듈 내 구성요소들이 연관성이 있고, 특정 순서에 의해 수행
시간적 응집도	모듈 내 구성요소들이 서로 다른 기능을 같은 시간대에 함께 실행
논리적 응집도	모듈 내의 구성 요소들이 논리적으로 연관된 임무나 비슷한 기능을 수행
우연적 응집도	모듈 내부의 구성요소들이 서로 관련이 없음

응집도 중에서 기능적 응집도는 가장 높은 응집도를 가지고, 우연적 응집도가 가장 낮은 응집도를 갖는다. 응집도가 높을수록 한 모듈 내의 함수와 데이터들의 연관관계가 높기 때문에 변경, 유지보수에 용이해지고, 품질이 우수해짐을 의미한다.

3. 예시 코드 비교

절차식 언어와 객체지향 언어에서 응집도 측정을 위해서는 먼저 모듈을 정의 한다. 절차식 언어에서는 기능의 최소 단위인 Function을, 객체지향 언어에서는 Class를 모듈로 정의한다. 절차식 언어 및 객체지향 언어에서 정의된 단위들을 도형으로 표현한 결과는 그림 2와 같다.

(그림 2) 구현 가시화를 위한 역할별 도형 정의

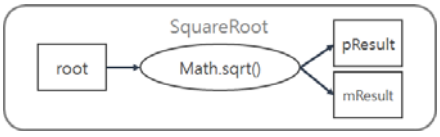
3.1 기능적 응집도

절차식 언어에서는 하나의 함수 안에서 변수선언과 조건문 연산이 변수값을 계산하는 하나의 기능을 수행 할 경우, 객체지향 언어에서는 표 2와 같이 하나의 클래스 안에서 변수선언과 Math 객체를 통한 연산이 제곱근을 구하는 하나의 기능을 수행하였다.

<표 2> 객체지향 언어의 기능적 응집도 소스코드 및 그래프

객체지향 언어

```
public class SquareRoot{
    public static void main(String[] args) {
        int root = 9;
        double pResult, mResult;
        pResult = Math.sqrt(root);          //양의 제곱근
        mResult = -Math.sqrt(root);         //음의 제곱근
        System.out.println("9의 제곱근 : " + pResult + mResult:);     }
```



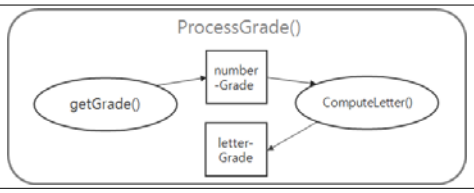
3.2 순차적 응집도

절차식 언어에서는 한 함수 안에서 변수 선언, 다른 함수로부터 받은 return값을 또 다른 함수에 인수로 전달하는 경우, 객체지향에서는 표 3과 같이 한 클래스에서 여러 메소드를 생성하고 한 메소드에서 나온 출력 값을 다른 메소드에 입력 하였다.

<표 3> 객체지향 언어의 순차적 응집도 소스코드 및 그래프

객체지향 언어

```
public class ProcessGrade {
    public int getGrade(){
        ...
        return numberGrade; }
    public String computeLetter(int _numberGrade){
        ...
        return letterGrade; }
    public void displayLetter(String _letterGrade){
        System.out.println(_letterGrade); }
    public static void main(String[] args) {
        int numberGrade;
        String letterGrade;
        ...
    }
}
```



3.3 교환적 응집도

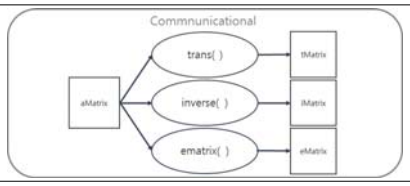
절차식 언어에서는 하나의 함수에서 이차원배열 선언과 같은 입력 값을 이용해 여러개의 다른 함수로부터 각각의 계산을 수행할 경우, 객체지향에서는 표 4와 같이 선언된 하나의 변수를 같은 클래스 내의 다른 메소드에 전달하여 각기 다른 결과를 출력한다.

<표 4> 객체지향 언어의 교환적 응집도 소스코드 및 그래프

객체지향 언어

```
public class Communicational {
    void ComputeMatrix(int transformMatrix[][], int verseMatrix[][],
int eMatrix[][]){
        int[][] aMatrix={ {0},{0} };

        for(int i=0;i<5;i++){
            for(int j=0;j<5;j++){
                aMatrix[i][j] = i+j;
            }
        }
        transformMatrix = trans(aMatrix);
        inverseMatrix = inverse(aMatrix);
        eMatrix = ematrix(aMatrix);
        }
        int[][] trans(int[][] aMatrix){ return tMatrix;}
        int[][] inverse(int[][] aMatrix){ return iMatrix;}
        int[][] ematrix(int[][] aMatrix){ return eMatrix;}
}
```



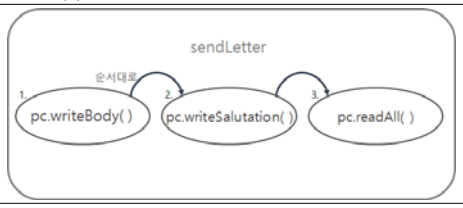
3.4 절차적 응집도

절차적 언어는 절차에 맞게 순서대로 기능들을 수행하지만, 하나의 함수에서 다른 함수들을 호출하여 기능을 수행한 경우, 객체지향에서는 한 클래스 안에서 객체를 생성하고 각각의 메소드를 절차적으로 수행하였다. 절차적 응집도에서 소스코드와 그래프의 예는 표 5와 같다.

<표 5> 객체지향 언어의 절차적 응집도 소스코드 및 그래프

객체지향 언어

```
public class sendLetter {
                public void writeBody() throws IOException{ ... }
                public void writeSalutation(){ ... }
                public void readAll() throws Exception{ ... }
                public static void main(String[] args) throws Exception {
                        Procedural pc = new Procedural();
                        pc.writeBody();
                        pc.writeSalutation();
                        pc.readAll();
                } }
```



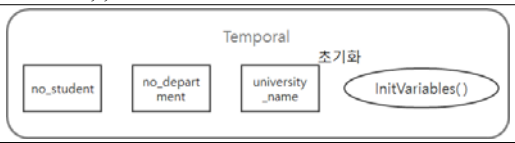
3.5 시간적 응집도

절차적 언어에서는 한 함수 내에서 선언과 초기화하는 기능을 같은 시간대에 한꺼번에 수행 한 경우, 객체지향에서는 표 6과 같이 한 클래스 내에서 전역 변수로써 변수들을 선언하고, 또한 클래스 내의 메소드 안에서 변수들을 초기화 시키는 기능을 수행하였다.

<표 6> 객체지향 언어의 시간적 응집도 소스코드 및 그래프

객체지향 언어

```
public class Temporal {
                int noStudent, int noDepartment;
                String universityName;
                void InitVariables(){
                        noStudent = 0;
                        noDepartment = 0;
                        universityName = "Hongik University";
                } }
```



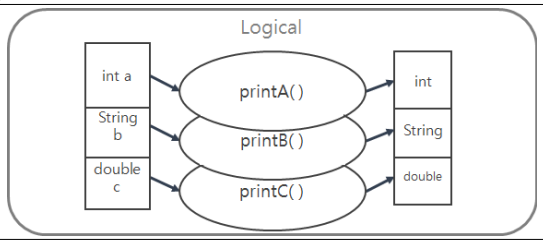
3.6 논리적 응집도

절차적 언어에서는 한 함수 내에서 논리적으로 연관되고 비슷한 연산기능을 수행하는 경우, 객체지향에서는 표 7과 같이 한 클래스 안에서 생성된 여러 메소드들이 각각 다른 변수를 인자로 받아 각각 출력하는 기능을 하였다.

<표 7> 객체지향 언어의 논리적 응집도 소스코드 및 그래프

객체지향 언어

```
public class Logical {
        void printA(int _a){ System.out.println(_a); }
        void printB(String _b){ System.out.println(_b); }
        void printC(double _c){
                _c++;
                System.out.println(_c);
        }
        public static void main(String[] args) { ... }
}
```



3.7 우연적 응집도

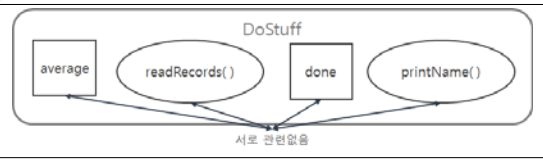
표 8과 같이 객체지향 언어에서 기능적으로 전혀 관련 없는 변수선언, 함수 호출, 연산기능 또는 객체 생성, 메소드 생성 등 을 수행하였다. 이로써 우연적 응집도에서는 서로 전혀 관련 없는 기능들이 모여있으므로 비교가 무의미 하다.

<표 8> 객체지향 언어의 우연적 응집도 소스코드 및 그래프

객체지향 언어

```
public class DoStuff {
        void readRecords(){
                int record;
                Scanner sc = new Scanner(System.in);
                record = sc.nextInt();
        }
        void printNames(){
                System.out.println("이름");
        }
        public static void main(String[] args) {
                int average;
                int totalScore = 10;
                Coincidental cc = new Coincidental();

                cc.readRecords();
                average = totalScore/10;
                cc.printNames();
        }
}
```



종합하면, 절차식 언어의 순차적, 교환적, 절차적 응집도, 이 3개의 응집도 예시코드에서 나타난 공통점으로는 하나의 모듈안에서 완전한 기능을 수행하지 못했다. 만약, 하나의 모듈안에서 모든 수행을 하도록 수정하면 그 결과 모듈의 응집력이 약해진다. 반면, 객체지향 언어에서는 기능적 응집도에서 계산하는 메소드를 만들어서 수행해도 되는 기능이지만, Math객체를 사용하고 각 클래스 안에서 모든 기능들이 수행되었다.

4. 결론 및 향후연구

본 논문에서는 절차식 파라다임과 객체지향의 응집도를 모듈 단위에 따라 내부 소스코드를 제시하고 그래프로 가시화하여 비교 분석하였다. 응집도를 이용한 코드복잡도를 가시화 하면 소스코드의 근본적인 오류 검출에 용이해지며, 복잡도를 줄일 수 있게 된다. 때문에, 이 연구를 통해 응집도를 SW Visualization을 통해 가시화하여, 코드복잡도를 표현하는데에 적용할 예정이다.

ACKNOWLEDGEMENT

이 논문은 2015년 교육부와 한국연구재단의 지역혁신창의 인력양성사업의 지원을 받아 수행된 연구임(NRF-2015H1C1A1035548)

참고문헌

[1] Daniel Rodriquez, Rachel Harrison, “An Overview of Object-Oriented Design Metrics”, 2001. 03.

[2] 권하은, 손현승, 서채연, 김영수, 박병호, 김영철, “기존 모듈 간의 결합도 및 응집도 개념과 객체지향 파라다임과의 관련 비교 연구”, 한국정보과학회, 2014, 06.

[3] 권하은, 박보경, 이근상, 박용범, 김영수, 김영철, “코드 가시화부터 모델링 추출을 통한 역공학 적용”, 한국정보처리학회, 제21권, 제2호, 2014. 11.

[4] 이종석, 우치수, “객체 지향 시스템에서의 클래스 응집도와 결합도 메트릭”, 정보과학회논문지 : 소프트웨어 및 응용 27(6), 2000.6, 595-606

[5] Johann Eder, Gerti Kappel, Michael Schre, “Coupling and Cohesion in Object-Oriented Systems“, 1992