프로그램에 대한 복잡도 측정인자 분석

An Analysis of the Complexity Measurement Factor for a Program

이 규 범* 송 정 영**
Kyu-Boem Lee Jeong-Young Song

요 약

객체지향 프로그램에 대한 복잡도 측정에 관한 방법으로서 기존의 객체지향 패러다임 특성이라 할 수 있는 객체, 메시지, 클래스, 캡슐화 및 상속성등에 관한 연구는 이미 보고된바 있다. 본 연구에서는 측정인자로서 Halstead의 Program Volume, Program Level, Program Impurity, Macabe의 Cyclomatic Number, Handerson-Sellers의 응집도 결핍척도, Sullivan's의 PVG측정 방법 등을 선택하였다. 시뮬레이션으로는 객체지향 프로그램의 대표적인 언어인 JAVA Program을 Design & Coding할 때 도움을 주는 척도에 대해서 JAVA Sample Program을 준비하여 프로그램 복잡도 면에서 측정인자들을 각각 비교·분석 하였다.

Abstract

Measurement of the objects, messages, classes, capsulation, inheritance, etc. that are conventional object-oriented paradigm characteristics as a method of measurement of the complexity of object-oriented programs has been reported. In this paper, the measures that are helpful to designing and coding of JAVA program, which is the representative language of object-oriented programs, are applied to six measures(i.e., Halstead's Program Volume, Program Level, Program Impurity, Macabe's Cyclomatic Number, Handerson-Seller's lack of cohesion in method and Sullivan's PVG.) suggested in the present study by referring to several actual programs as example for comparative analysis.

1. 서 론

일반적으로 S/W를 크게 분류하면 시스템 소프트웨어, 실시간 소프트웨어, 비즈니스 소프트웨어, 공학·과학 소프트웨어, 내장용 소프트웨어, PC 소프트웨어, 인공지능 소프트웨어로 분류된다. 이러한 소프트웨어를 이해하기 위해서는 소프트웨어의 특성을 조사해 보는 것이다[2]. 소프트웨어는 물리적인 시스템 요소라기보다는 논리적인 요소이다. 그러므로 소프트웨어는 하드웨어특성보다 상당히 다른 특성을 가지고 있는데, 소프트웨어는 개발되는 것이고 공학화 되는 것이다. 소프트웨어와 하드웨어의 제조 사이에는 유사성이 존재하지만 활동은 기본적으로 다르다. 소프트웨어의 이용은 설계에 집중되어 있다. 그리고 소프트웨어를 사용하면 없어지는 것이 아니고, 완성된 후 결함이 발견되면 수정 보완함으로서 계속 사용할 수 있는 것이다[6]. 그러므로 완성된 S/W에 대한 올바른 품질평가 기준은 수정·보완하는데 있어서 매우 중요한 정보로 사용된다. 이러한 품질평가 기준은 보는 관점에 따라 그 중요도에 있어서 현저한 차이가 있을 뿐만 아니라 또한 정량적 평가기준과 정성적 평가기준에 따라서도 그 정의가 불분명 한 것이 현재의 사실이다. 또한 응용 S/W를 설계·개발·구현하는데 있어서 사용자 입장에서의 충분한 검토가 필요하다[6].

1.1 소프트웨어 품질척도 기준

소프트웨어의 품질이란 품질을 위해서 사용자가 있고 품질의 실행이 자연적 성향에 따라서 움직이는 것과 같이 품질이 자연스럽게 사용되는데 의미가 있다. 그러나 완벽한 품질이 존재하지 않

* 정 회 원 : 배재대학교 컴퓨터공학과 박사과정
wsclkb7@mail.pcu.ac.kr
** 종신회원 : 배재대학교 컴퓨터공학과 교수
jysong@mail.paichai.ac.kr

다고 하더라도 정확한 표현, 기술, 연산, 추정이 요구되는데 특별히 소프트웨어의 품질은 기능과 성능, 사용자의 요구사항들을 명확히 나타내고 개발 표준을 명확하게 문서화시키며, 전문적으로 개발된 소프트웨어에 기대되는 특성을 암시해 줄 수 있도록 해야한다. 그러기 위해서는 소프트웨어의 요구사항이 품질을 측정하는 기초가 되며, 명시된 표준들이 소프트웨어를 공학화 시키는 방법을 안내하고, 자주 언급되지 않는 암시적인 요구들의 집합을 마련해야 한다. 소프트웨어의 품질을 평가하기 위해서는 직접적으로 측정 가능한 정량적인 측면과 간접적으로 측정 가능한 정성적인 측면으로 분리하여 정의 가능하다.

1.1.1 정성적 기준

주어진 소프트웨어에 대하여 직접적으로 평가할 수 없는 기준 척도로서 다음과 같이 정의할 수 있다.

첫째, 소프트웨어의 전체적인 품질의 중요한 요소로서 소프트웨어의 신뢰성을 들 수 있다. 소프트웨어가 실행되는데 있어서 반복적으로 실패를 거듭한다면 의심이 가는 품질로 받아들일 수밖에 없다. 즉, 사용자의 요구되는 정밀도로 의미한 기능이 수행되기를 기대하는 정도이다. 둘째, 안정성을 들 수 있다. 안정성에 대한 소프트웨어에 부정적으로 영향을 주는 것으로 소프트웨어 전체시스템에 실패의 원인을 제공하는 잠재적인 위험과 식별 평가에 주의를 해야한다. 무기 시스템, 항공기 운항제어, 원자로 등과 같이 안전 정밀과정을 제어하는 시스템에서의 안전성 결함이란 엄청난 경제적 손실 또는 최악의 인명 사상자를 자아내게 할 수 있기 때문이다. 셋째, 정확성으로 개발된 소프트웨어는 정확하게 작동되어야 한다. 그렇지 않을 경우 사용자에게는 거의 가치 없는 것이 되고 말기 때문이다. 정확성은 소프트웨어가 요구된 기능들을 수행하는 정도이다. 정확성에 대한 가장 일반적인 측정은 결함의 정도이다. 결함이란 요구사항에 일치하지 않은 경우를 말한다. 넷째, 유지보수성이다. 소프트웨어 공학 입장에서 본 유지보수성은 다른 어떤 공학 활동보다 더 많은 노력을 요구한다. 유지 보수성은 소프트웨어의 오류가 발생되면 수정할 수 있고 프로그램의 환경이 변하면 환경에 적응시킬 수 있고, 사용자가 요구사항을 변경하면 이에 맞출 수 있는 소프트웨어의 용이성을 말한다. 그러나 유지보수성을 직접 측정할 수 있는 방법은 아직 존재하지 않는다. 간접적인 측정방법으로는 MTTC(Mean-Time-To-Change)로서 변경하는데 걸린 평균시간이다. 여기에서 시간이란 변경요구를 변경하고, 분석하고, 적절한 수정을 설계하고, 변경을 구현하고, 테스트하며 변경된 것을 모든 사용자에게 분배하는데 걸린 시간이다. 다섯째, 무결성 이다. 이것은 해커와 바이러스 시대에서는 더욱 중요한 요소로 대두된다. 이 속성은 시스템의 보호를 위하여 공격에 저항하는 시스템의 능력이다. 이러한 공격은 프로그램, 데이터, 문서 모두에게 발생될 수 있다. 무결성을 측정하기 위해서는 위협과 보안이라는 추가 속성이 먼저 정의되어야 한다. 위협은 특수한 유형의 공격이 주어진 시간 내에 발생하는 확률로 나타낼 수 있으며, 보안은 특수한 유형의 공격을 물리칠 수 있는 확률로서 어느 정도는 측정될 수 있다. 여섯째, 사용성 이다. 소프트웨어 제품을 논의할 때마다 나타나는 말은 "사용자의 친숙성"이다. 소프트웨어가 친숙성이 결여되어 있다면 그 소프트웨어를 수행하는 기능이 가치 있는 것이라 할지라도 그 소프트웨어는 자주 실패하기 때문이다.

1.1.2 정량적 기준

개발된 소프트웨어에 대하여 정량적으로 평가하기 위한 기준은 주어진 소프트웨어를 직접적으로 평가할 수 있어야 한다.

첫째, 소프트웨어의 성능으로 처리속도, 응답시간, 처리능력으로 측정 가능하다. 처리 속도를 측정하기에는 소프트웨어의 통합단계로서 측정가능

한데 소프트웨어 통합단계에는 다음과 같은 4가지 방법으로 측정 가능하다.

- 구조 테스트 측정 : 처리 흐름 도표를 사용하여 입력 클래스와 관련된 작업으로 입력의 처리와 결과의 출력으로 구성된다.
- 기능 테스트 측정 : 사용자의 요구분석에 나타난 사항이 제대로 구현되었는가를 측정하기 위한 것으로 시스템 전체가 하나의 블랙박스로 간주된다.
- 성능 테스트 측정 : 여러 기능을 수행하는데 소요되는 시간을 시스템이 정상적인 상태에서 처리하는가, 과부하인 경우에도 많이 자제하지 않고 처리하는가를 측정하는 것이다.
- 스트레스 테스트 측정 : 과다한 입력이 주어졌을 때 시스템이 어떻게 처리하는가를 측정하는 것으로 시스템 반응 시간을 점검하는 것도 포함된다.

둘째, 소프트웨어를 확장할 수 있는가, 바로 적응 가능한가, 실용성이 있는가 등의 유지 보수성이 있는가, 그리고 메뉴나 윈도우가 적합하게 구성되어 있는가, 구성은 사용자가 보기 쉽게 배치되어 있는가 등이다.

셋째, 개발비용 또한 여타 관계 있는 소프트웨어 또는 비슷한 이미 개발되어 있는 소프트웨어와 비교해 볼 때 적당한 비용이 산출되어 있어야 한다. 개발비용 산정시 고려해야할 요소로서는 인적자원, 하드웨어자원, 소프트웨어자원, 개발기간 등으로 볼 수 있다. 소프트웨어 자원 기법으로 원시코드 라인을 산출하는 기법으로 PERT(Project Evaluation and Review Technique)의 예측 공식을 이용할 수 있다.

넷째, 이론적으로는 특정한 알고리즘에 대하여 최소한의 Size로 존재해야 한다는 면에서 프로그램 전체길이 N과 프로그램 volume V를 Halstead는 각각 식 (1.1)과 식 (1.2)로 정의했다[1].

$$N = n_1\log_2 n_1 + n_2\log_2 n_2 \quad (1.1)$$

$$V = N\log_2(n_1 + n_2) \quad (1.2)$$

여기에서, n_1 은 서로 다른 operator의 수, n_2는 서로 다른 operand의 수, N_1은 operator 전체 수, N_2는 operand 전체 수를 나타내고 있다.

다섯째, 또한 주어진 알고리즘에 대하여 얼마나 복잡하게 디자인되어 있는지를 매트릭스로 측정 가능하다[2]. 이것은 프로그램을 그래프로 제어 흐름을 묘사할 수 있다[3]. 일반적으로 측정 가능한 매트릭스 표현은 식 (1.3)과 같다. McCabe는 프로그램 제어흐름에 관하여 수학적 근본의 바탕위에 G(Graphic)의 Cyclomatic 수 V(G)를 복잡도를 정의하였다.

$$V(G) = e - n + 2p \quad (1.3)$$

여기에서, e는 Edge의 수, n은 Node의 수, p는 연결성분의 수를 나타내고 있다.

V(G)의 수치가 크면 클수록 프로그램의 제어흐름이 복잡하다는 것을 나타낸다.

이 복잡도는 프로그램의 외형적인 크기나 문장 배열 순서와는 무관하며 단지 제어 흐름의 수에 의하여 결정되는 문제점이 있다.

2. 복잡도 척도

소프트웨어 품질 척도는 소프트웨어가 가지는 품질 특성들을 객관적이고 정량적으로 측정하기 위한 척도이다. 본 장에서는 소프트웨어가 가지고 있는 품질 특성들과 이들 품질 특성들을 측정하기 위한 사용자 중심척도 그리고 이해도와 신뢰도 등과 같은 품질 특성을 예측하기 위해 설명변수로 사용하는 소프트웨어 복잡도에 대해 살펴보기로 한다.

2.1 복잡도 측정 인자들

(1) Program Volume

프로그램 볼륨은 다음 식 (2.1)과 같이 정의한다.

$$V = N\text{Log}_2 n \quad (2.1)$$

여기에서, n_1은 연산자의 종류로서 실행 가능한 문장, 콤머, 라이브러리함수 등이 있고, n_2는 피연산자의 종류로서 변수, 상수 등이 있으며, N_1은 연산자의 전체 출현횟수, N_2는 피연산자의 전체 출현횟수로 $n = n_1 + n_2$, $N = N_1 + N_2$를 나타내고 있다.

(2) Program Level

프로그램 레벨은 다음 식 (2.2)와 같이 정의한다.

$$L = (2 / n_1) * (n_2 / N_2) \quad (2.2)$$

(3) Complementary Operations-Impurity

Complementary Operations-Impurity는 다음 식 (2.3)과 같이 정의한다.

$$V \times L = (N_1+N_2)*\log_2*(n_1+n_2) * \frac{2}{n_1} * \frac{n_2}{N_2} \quad (2.3)$$

여기에서, n_1은 유일한 연산자의 개수, n_2는 유일한 피연산자의 개수를 나타낸다.

(4) Cyclomatic Number

V(G)의 수치가 크면 클수록 프로그램의 제어흐름이 복잡하다는 것을 나타낸다. 그 식은 식 (2.4)에서 정의한다.

$$V(G) = e - n + 2p \quad (2.4)$$

여기에서, e는 에지(Edge)의 수, n는 노드(Node)의 수, p는 연결성분(Connected Componet)의 수를 나타낸다.

(5) Lack of COhesion in Method

클래스의 응집도 결핍을 측정하는 것이다. 한 클래스의 응집도는 메소드들이 클래스내의 인스턴스 변수와 얼마나 밀접히 관련되어 있는지에 의해 특징 지워진다. 클래스가 응집력이 크면 클수록 그 클래스를 유지보수 하기가 더욱 쉽기 때문에 응집도 결핍척도가 크면 클수록 그 클래스를 유지보수하기가 점점 더 어려워진다는 것을 알 수 있다.

응집도 결핍은 Henderson-Sellers가 정의한 식을 사용하였다. 단, 응집도 결핍의 값은 0에서 2까지의 값을 가진다. 다음 식 (2.5)와 같이 정의한다.

$$LCOM = \frac{\frac{\text{전체변수 개수}}{\text{변수 개수}} - \text{메소드 개수}}{1 - \text{메소드 개수}} \quad (2.5)$$

(6) Process V(G)

이 복잡성 척도는 sullivan의 P2척도에 있어서 C2 대신에 McCabe의 V(G)를 이용함으로써 유도되었다.

$$PVG = \sum_i V(Gd_i) = \sum_i (e_i - n_i + 2) \quad (2.6)$$

여기에서, i는 프로그램의 데이터 아이템수, d_i Σ는 프로그램에서 나타나는 데이터 아이템, Gd_i는 Nd_i, Ed_i, n_s, n_t, Nd_i는 $N(d_i) + \{n_s, n_t\}$, $N(d_i)$는 d_i,를 나타내는 노드(node)수, n_s는 프로그램의 시작노드, n_t는 프로그램의 종단노드, Ed_i는 $\{(n_j, n_k)$: n_k는 Nd_i에 포함되지 않는 edge에 의해 n_j로부터 도달 가능한 노드$\}$, e_i는 Gd_i의 에지(edge)수, n_i는 Gd_i의 노드(node)수를 각각 나타낸다.

2.2 분류트리 형성

객체들을 특정한 범주들로 분류하기 위해서 분류 트리를 이용한다. 분류 트리는 단말 노드와 비단말노드로 구성된다. 단말 노드는 하나의 범주로 분류가 완료된 상태를 나타내고, 비단말 노드는

```
import java.awt.*;
import java.applet.Applet;

public class CheckboxButton extends Applet {
    public void init() {
      add(new Checkbox("Cake"));
      add(new Checkbox("Ice cream"));
      add(new Checkbox("Pie"));
    }
    public boolean action (Event evt, Object arg) {
      if (evt.target instanceof Checkbox) {
          handleCheckbox(evt,arg);
          return true;
      }
      return false;
    }

    public void handleCheckbox(Event evt,Object arg) {
      System.out.print(((Checkbox)evt.target).getLabel());
      System.out.println("=" + arg);
    }
}
```

척 도	n1	n2	N1	N2	n	N	V	L	I	V(G)	LCOM	PVG
척도값	22	4	57	12	26	69	324.33	0.030	324.3	2	0.778	1

(그림 1) JAVA Program 척도의 적용예

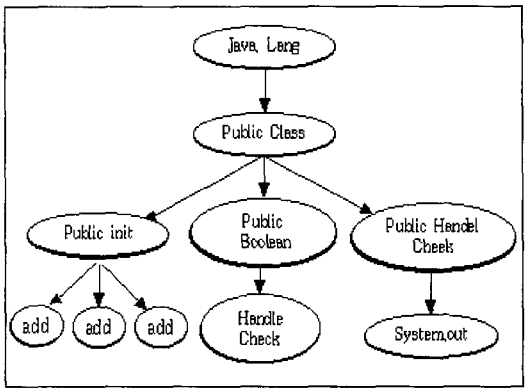
하나의 객체들이 여러 범주에 속하기 때문에 다른 속성을 이용하여 테스트를 더해야 분류가 완료되는 것을 나타낸다.

그 프로그램이 구조적으로 형성된 모양을 기본으로 하위그룹과 상위그룹의 분류 트리를 형성하였다. 그림 1은 간단한 JAVA 프로그램에 대한 대표적인 척도들의 측정값을 보여 준다. 그림 1에 대한 분류 트리를 형성시켜 본 결과를 그림 2에 보이고 있다.

3. Sample Program의 평가

3.1 JAVA Sample Program 선택

본 연구에서는 program의 복잡도를 측정하기 위하여 앞에서 서술한 6가지 척도를 선택하여 비교·분석을 행하였다. 그 중에서 program들은 ImputOut program, ChracterInput program, Fileoutput program, do-while program, YesORNo program, CheckBoxButton program, ListDemo program, TextAreaDemo program, Scroll program, GridBagTest program, Draw Rectangle program등으로 Sample program을 선정하였다. 이들 program들은 대략 라인 코드수가 30~150 line number를 가지고 있다.



(그림 2) 그림 1의 program에 대한 결정 트리의 예

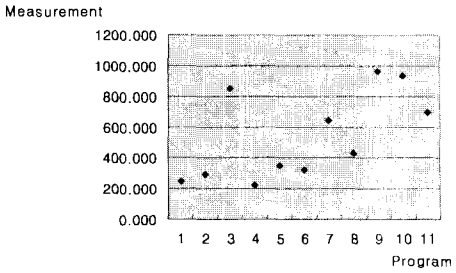
3.2 분석 및 평가

위 2장에서 서술한 complexity measurement들에 대해서 각 sample program에 측정한 결과치가 표 1과 같이 나타나 있음을 확인하였다.

본 연구에서 복잡도 측정인자로 고려된 Program Volume, Program Level, Complementary Operations-Impurity, Cyclomatic Number에 대하여 주어진 식에 대입하여 시뮬레이션을 행한 결과 Program Volume에 대해서는 최소 222부터 최대 962를 갖는 값으로 분포가 전체적으로 산재되어 있음을 알 수 있고, Program Level에 대해서는 대체적으로 0.05이하의 값을 갖고 있는 것을 알 수 있으며, Complementary Operations-Impurity에 대해서는 선정된 Sample program 각각에 대해서 골고루 분포되어 있음을 알 수 있으며, Cyclomatic Number에 대해서는 전체적으로 양극화 현상이 나타나고 있음을 볼 수 있다. 각각의 인자요소에 대하여 구체적으로 그림으로 나타내보면 다음과 같다.

(1) Program Volume

대체적으로 Program Volume이란 식 (2.1)에서 보여준 바와 같이 연산자와 피연산자 수의 출연



(그림 3) Volume of sample program

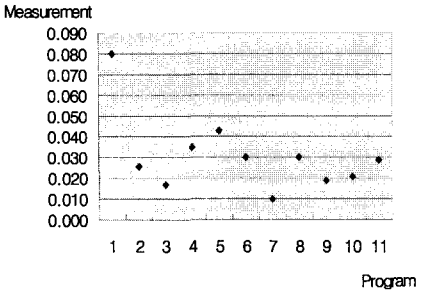
횟수에 비례함으로 인하여 JAVA Sample Program 3번, 9번, 10번보다 1,4번 Program이 현저히 낮음을 알 수 있다. 표 1에서 보여준 Program Volume에 대한 각각의 수치를 비교하기 쉽도록 표현해 보면 그림 3과 같다.

(2) Program Level

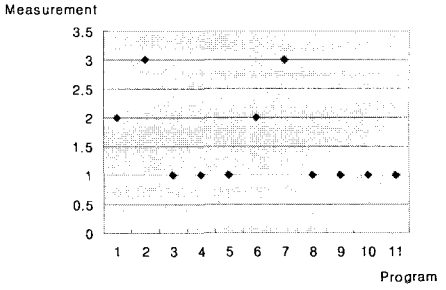
식 (2.2)에서 나타난 바와 같이 Program Level이란 일반적으로 N_1과는 무관하며 n_2와 비례, N_2와는 반비례하고 있음으로 표 1의 1번 프로그램에서는 비교적 다른 프로그램들보다는 현저히 큰 값을 가지고 있음을 알 수 있다. 그 외에 다른 Sample 프로그램들은 서로 많은 차이가 나지 않는 값을 가지고 있는 것을 그림 4에서 나타내고 있다.

(표 1) JAVA sample program measurement

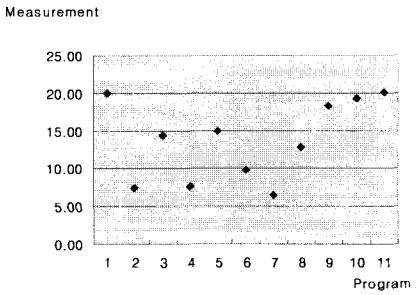
#	Measurement / Program	n1	n2	N1	N2	n	N	V	L	I	V(G)	LCOM	PVG
1	InputOut	25	9	40	9	34	49	249.286	0.080	249.29	2	0.556	2
2	ChrInput	26	2	55	6	28	61	293.249	0.026	293.25	3	0.600	3
3	FileOut	38	10	121	31	48	152	848.914	0.017	848.91	1	0.753	1
4	Do-While	23	2	43	5	25	48	222.905	0.035	222.91	1	0.250	3
5	YesOrNo	31	2	66	3	33	69	348.063	0.043	348.06	1	0.917	2
6	CheckboxButton	22	4	57	12	26	69	324.330	0.030	324.33	2	0.778	1
7	ListDemo	29	7	76	48	36	124	641.071	0.010	641.07	3	0.697	1
8	TestAreaDemo	24	9	60	25	33	85	428.774	0.030	428.77	1	0.600	1
9	Scroll	45	18	119	42	63	161	962.342	0.019	962.34	1	0.887	1
10	GridBagTest	27	19	101	68	46	169	933.482	0.020	933.48	1	0.250	1
11	DrawRectan	22	15	86	47	37	133	692.857	0.029	692.86	1	0.741	1



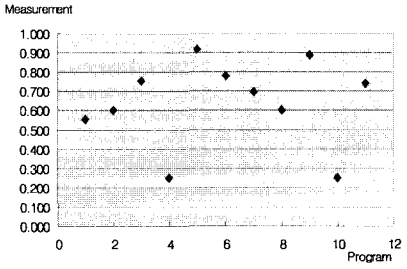
(그림 4) Level of sample program



(그림 6) V(G) of sample program



(그림 5) Impurity of sample program



(그림 7) LCOM of sample program

(3) Complementary Operations-Impurity

Complementary Operations-Impurity는 식(2.3)에서 보여주는 바와 같이 Program Volume과 Program Level의 관계식으로 인하여 1번, 10번, 11번 Program은 비교적 큰 수치를 나타내고 있고, 6번 Program은 비교적 낮은 수치를 나타내고 있다. 그리고, 이러한 관계를 구체적으로 보면 그림 5에 나타낸 바와 같다.

(4) Cyclomatic Number

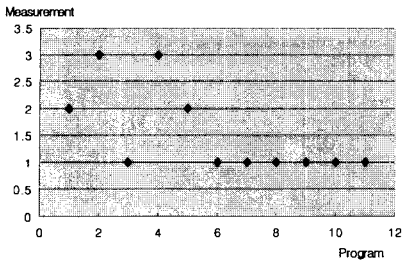
Cyclomatic Number는 식 (2.4)에서 나타난 바와 같이 V(G)의 값이 클수록 제어흐름이 복잡해지고 그 Program이 얼마나 구조화 되어있는 가를 판단 가능하게 함을 알 수 있다. 그림 6에서 보는 바와 같이 Sample Program 6번은 일반적인 다른 프로그램들보다 비교적 큰 값을 가지고 있음을 알 수 있다.

(5) LCOM

측정척도 LOCM의 각 Sample program에 대한 추이는 그림 7에서 보는 바와 같다. 그림 7에서 보는바와 같이 do-while program 과 GrideBagTest program의 경우에는 0.3이하의 작은수를 Mark하고 있다. 이는 Class 응집도 결핍이 method내 Instance Variable과 별로 관련이 없음을 보이고 있다. 반면 YesORNo program과 Scroll program은 강한 관계를 가지고 있음이 확인되었다.

(6) PVG

측정척도 PVG의 각 Sample program에 대한 추이는 그림 8에서 나타내는 바와 같다. 그림 8에서 보는 바와 같이 입출력 program, 문자 program, 화일출력 program, do-while program, YesORNo program은 일정한 Rule이 없이 산재되어 있음을 알 수 있다. 그러나, CheckboxButton program, ListDemo program, TextAreaDemo program, Scroll program, GridBagTest program, 사각형그리기 program은 동일한 값인 1.0을 Mark하고 있음을 알 수 있다. 이



(그림 8) PVG of sample program

는 특히 객체지향 program인 JAVA에서 윤곽을 나타내는 척도인 PVG가 일정함을 알 수 있다.

4. 결 론

본 연구에서는 객체지향 프로그램의 대표적인 언어인 JAVA Program을 Design & Coding할 때 도움을 주는 척도에 대해서 6가지 척도를 선택하여, 11개의 sample program에 대한 분석을 하였다.

그 결과 JAVA Program에 대한 원시코드를 작성할 때 Complexity 관점에서의 Program Volume, Program Level, omplementary Operations- Impurity, Cyclomatic Number의 추이와 특성을 나타내 보였다. 향후 연구계획으로는 좀더 많은 Sample program을 준비하여 다방면에서 분석 조건을 제시하여 Complexity 면에서 Simulation을 행하여 보는 것과, 일 만개 Source Code Line Number 이상의 program에도 적용시켜 각각의 측정인자에 대하여 비교해 보는 것이 형평성을 잃지 않을 것으로 생각된다.

참 고 문 헌

[1] M. Halstead, "Element of Software science," Elsevier, North-Holland, 1977.

[2] T.McMcab, "A complexity measure," IEEE tran. on Software Engineering, Vol.2, No.4, pp. 308~320, Apr. 1976.

[3] W. Curitis, et al., "Measuring the Phychological Complexity of software Maintenance Tasks with the Halstead and McCabe Metrics," IEEE tran. on Software Engineering, Vol.5, pp. 96~104, Mar. 1973.

[4] John D. Musa, Anthony Iannio, Kaznhira Okumoto, "Software Realiability : Measurement, Prediction, Application," McGraw-Hill. 1987.

[5] Ken S. Lew, Tharm S. Dillon, Kevin E. Forword, "Software Complexity and Its Impact on Software Reliability," IEEE Trans. on SE, Vol.14, No. 11, Nov. 1988.

[6] S. D. Conte, H. E. Dunsmore, V. Y. Shen, "Software Enginneering Metrics and Models," The Benjamin/Cummings Publishing Company, Inc., 1985.

[7] John Stephen Davis, Richard J. Leblanc, "A Study of the Applicability of Complexity Measures," IEEE Trans. on SE, Vol. 14, No. 9, Sep. 1988.

[8] Mohammad A., "Putting Metrics into Software Perspectives," "http://www.cs.usa.ca/hompages/grads/moal135/856/metrics/meteics.html," Oct.1995.

[9] Lee Kyu-Boem and Song Jeong-Young, "An Analysis for Hangul 97 and MS-word 97(The Consideration on User Side)," Electronics, Information and Systems Society, I.E.E. of Japan, pp. 653~656, Step. 2000.

[10] Lee Kyu-Boem and Song Jeong-Young, "Measures for Complexity Measurement of JAVA," Electronics, Information and Systems Society, I.E.E. of Japan, pp. I-551~I-554 Step. 2001.

[11] Henderson-sellers B., "Object-Oriented Metrics Measures of Complexity," Prentice-Hall, Hemel Hempstead. UK, 1996.

[12] Briand L., Morasca S., and Basili V., "Property Based Software Engineering Measurement," IEEE Trans on Software Engineering, Vol. 22, No. 1, pp. 68~86, Jan. 1996.

◐ 저 자 소 개 ◑

이 규 범

1987년 한남대학교 컴퓨터공학과 졸업(공학사)
1999년 한남대학교 대학원 컴퓨터공학과 졸업(공학석사)
2000년~현재 : 배재대학교 대학원 컴퓨터공학과 박사과정
관심분야 : 프로그램 복잡도, 시스템 소프트웨어, 소프트웨어공학, 자연어처리, Human-interface
E-mail : wsclkb7@mail.pcu.ac.kr

송 정 영

1984년 한남대학교 컴퓨터공학과 졸업
1992년 와세다대학교 전기전자정보공학연구과 졸업(공학석사)
1995년 동대학 박사과정 졸업(공학박사)
1995년~1997년 청운대학교 전자계산학과 전임강사
1997년~현재 : 배재대학교 컴퓨터공학과 교수
관심분야 : 문자·음성·영상처리, 시스템 소프트웨어, 소프트웨어공학, 자연어 처리, Human-interface.
E-mail : jysong@mail.paichai.ac.kr