Proceedings of KFIS 2001 Fall Conference, 2001. 12. 1

클래스 상속 구조의 유지보수성에 관한 척도

Metrics for Maintainability of Class Inheritance Structures

정 홍, 이재경
계명대학교 공학부

Hong Chung, Jae-Kyung Lee
School of Engineering, Keimyung University
jhong@kmu.ac.kr

요 약

본 논문은 Chidamber와 Kemerer가 제안한 객체지향 설계를 위한 척도를 바탕으로 이를 확장하여 클래스 상속 구조의 유지보수성을 이해성과 변경성 측면에서 측정하는 새로운 객체지향 척도를 제안했다. 그리고 클래스 상속 구조의 예를 들어 비교 평가를 함으로써 Chidamber와 Kemerer의 척도 및 Henderson-Sellers의 척도보다 우수함을 보였다.

Keyword: object-oriented software metrics, class, inheritance, maintainability

1. 서 론

소프트웨어 척도는 소프트웨어공학에서 소프트웨어의 복잡성과 품질을 측정하고 프로젝트의 노력과 비용을 추정하는데 있어서 필수적이다.

기능 점수(function point), 소프트웨어 과학(software science), 사이크로매틱(cyclomatic) 복잡도 등과 같은 고전적 척도는 절차적(procedural) 패러다임에서는 잘 사용되어 왔으나 클래스, 상속, 다형성(polymorphism) 등과 같은 객체지향 패러다임에는 잘 적용할 수가 없다. 따라서 10여 년 이상 여러 학자들이 객체지향 소프트웨어를 위한 별도의 척도가 필요한지, 필요하다면 무엇이 포함되어야 하는지를 논의해 왔다[9]. 초기에는 절차중심 프로그래밍에 사용되는 고전적 척도를 확장하는데 주력했으나[8,10], 최근에는 객체지향 프로그래밍을 위한 별도의 척도를 제안하고 있다[1,2,3,4].

대표적인 객체지향 척도에는 CK(Chidamber and Kemerer)[3,4]가 제안한 척도 집합이 있다. 클래스의 메소드 수를 측정하는 WMC(Weighted Methods per Class), 클래스의 조상 클래스 수를 측정하는 DIT(Depth of Inheritance Tree), 클래스의 서브클래스 수를 측정하는 NOC(Number Of Children), 클래스와 연결되는 다른 클래스의 수를 측정하는 CBO(Coupling Between Object classes), 클래스의 객체가 받는 메시지에 반응하여 실행되는 메소드의 수를 측정하는 RFC(Response For a Class), 프로그램간 상호 관련성을 측정하는 LCOM(Lack of Cohesion in Methods)이 그것이다.

하나의 척도 혹은 하나의 척도 집합으로 소프트웨어의 모든 특성을 측정하기에는 충분하지 못하다. 많은 객체지향 척도들 중 본 연구는 클래스 상속구조의 설계와 유지보수를 위한 척도에 초점을 맞추고자 한다. 왜냐하면 클래스 설계는 객체지향 시스템의 설계에 있어서 가장 우선 순위가 높으며, 상속은 객체지향 패러다임에서 가장 중요한 특성이기 때문이다. 따라서 본 연구에서는 객체지향 소프트웨어를 위한 새로운 척도인 클래스 상속 구조의 유지보수성 척도를 제안하고자 한다.

2. 관련 연구

클래스 상속 구조와 관련된 척도 중 CK[3]의 연구에 대해서 논한다.

2.1 DIT(Depth of Inheritance Tree)

클래스의 DIT 척도는 클래스의 상속 깊이로서, 다중상속인 경우 노드로부터 루트까지의 최대 깊이이다. DIT는 얼마나 많은 조상 클래스로부터 영향을 받는가를 측정한다. 그림 1의 클래스 상속

트리에서 A는 루트 클래스이므로 DIT(A)=0이며, B와 C는 루트로부터 거리가 각각 1이므로 DIT(B)=DIT(C)=1이다. 그리고 D와 E는 루트로부터 각각 최장 거리가 2이므로 DIT(D)=DIT(E)=2이다.

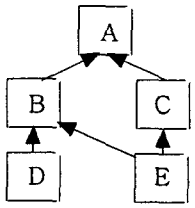

그림 1. 클래스 상속 트리

2.2 NOC(Number of Children)

NOC는 클래스 계층에서 클래스에 직접 종속된 서브클래스의 수이다. 이는 얼마나 많은 서브클래스가 부모 클래스의 메소드를 상속받는가를 측정하는 척도이다. 그림 1에서 A와 B의 자식이 각각 2이므로 NOC(A)=NOC(B)=2이며, C는 1이므로 NOC(C)=1이다. 그리고 D와 E는 자식이 없으므로 NOC(D)=NOC(E)=0이다.

DIT는 클래스가 조상 클래스의 특성에 의해 영향을 받는 범위를 나타내며, NOC는 자손에 대한 영향을 나타낸다. CK는 계층의 너비보다는 깊이를 권장하고 있다.

3. 클래스 상속 구조의 유지보수성 척도

클래스 상속 구조의 유지보수성을 정의하기 위해 그래프 이론의 용어를 사용한다. 활동을 정점으로 나타내고 선후관계를 간선으로 나타낸 그림 2와 같은 상속 구조에서 정점 i로부터 정점 j로의 경로가 있으면 정점 i는 정점 j의 선행자이고, 정점 j는 정점 i의 후속자이다[7]. 따라서 함수 PRED와 SUCC를 다음과 같이 정의한다.

PRED(i): 정점 i의 선행자의 총 수

SUCC(i): 정점 i의 후속자의 총 수

예를 들어 그림 2에서 PRED(E)=3, PRED(A)=PRED(B)=0, SUCC(C)=4, SUCC(G)=SUCC(H)=0이다.

시스템의 유지보수 활동에는 시스템의 구조를 이해해야 하는 이해 활동과 시스템을 수정해야 하는 변경 활동이 있다. 이해 활동에 있어서 시스템 이해성은 프로그램 구조나 클래스 상속 구조를 어느 정도로 쉽게 이해할 수 있는가 하는 것이며, 변경 활동에 있어서 시스템 변경성은 프로그램 구조나 클래스 상속 구조를 변경하기 쉬운 정도이다. 따라서 본 연구에서는 클래스 상속 계층의 유지보수성을 이해성 척도와 변경성 척도로 분리하여 제안한다.

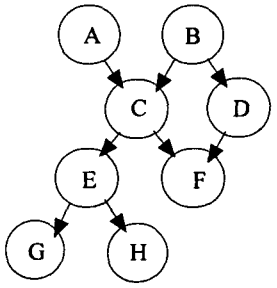

그림 2. 클래스 상속 구조

3.1 이해성 척도

이상적으로 볼 때 객체는 시스템을 구성할 때 재사용이 가능하고, 한 환경에서 다른 환경으로 이식하기 쉽도록 독립성을 가져야 한다. 그러나 실제로 객체들은 많은 상호 종속성을 갖고 있어 객체를 이해하고 변경하는데 어려움이 많다.

그림 2에서 F를 이해하려면 F 자체뿐만 아니라 상속을 하는 슈퍼클래스인 C와 D도 이해해야 한다. 또한 C를 이해하기 하기 위해서는 C의 슈퍼클래스인 A와 B를 이해해야 한다. 결과적으로 F, C, D, A, B의 5개 클래스를 모두 이해해야 하는데 이 값은 PRED(F)+1로 표시할 수 있다. 따라서 이해성 척도 U를 다음과 같이 정의한다.

$$U = PRED(Ci) + 1 \qquad (1)$$

여기서 Ci는 i번째 클래스이다. 그리고 클래스 상속 구조의 총 이해성 정도 TU는 다음과 같다.

$$TU = \sum_{i=1}^{t} (PRED(Ci) + 1) \qquad (2)$$

여기서 t는 상속 구조의 총 클래스의 수이다.

프로그램의 이해도나 복잡도를 거론할 때 규모가 큰 프로그램이 작은 것보다 복잡하다고 하는 것이 일반적이다. 그러나 프로그램간의 복잡도를 비교할 때는 어떤 기준, 예를 들어 같은 크기의 프로그램을 비교하는 것이 타당하다. 클래스 상속 구조의 복잡도도 마찬가지인데, 본 연구에서는 평균 개념 즉, 상속 계층의 평균 깊이를 적용하고자 한다. 상속 계층의 평균 깊이는 상속 구조에서 사용되는 일반적 모델링 수준이나 추상화 수준을 나타낸다[6]. 따라서 클래스 상속 구조의 평균 이해성 정도는 다음과 같다.

$$AU = \sum_{i=1}^{t} ((PRED(Ci) + 1)) / t \qquad (3)$$

그림 2의 예를 들면 AU=26/8=3.25인데 이는 클래스 상속 구조의 이해성 정도를 나타낸다.

3.2 변경성 척도

그림 2에서 클래스 C의 내용을 변경하려면 먼저 그것을 이해해야 하는데, C의 이해도는 U(C)이다. 만약 C가 서브클래스 E 혹은 F에 영향을 미친다면 이 또한 이해하여 변경해야 한다. 그리고 또 클래스 E가 서브클래스 G 혹은 H에 영향을 미친다면 이 또한 이해하여 변경해야 한다. 즉, 최악의 경우 C의 모든 후속자들을 변경해야 한다. 물론 최선의 경우에는 단지 C만 변경할 수도 있다. 다시 말하면, 후속자 서브클래스들중 반은 평균적으로 변경된다고 볼 수 있으므로 클래스 C의 변경성 정도는 U(C)+SUCC(C)/2라고 할 수 있다. 따라서 변경성 척도 M은 다음과 같이 정의한다.

$$M = U(Ci) + SUCC(Ci) / 2 \qquad (4)$$

여기서 Ci는 i번째 클래스이다.

그리고 클래스 상속 구조의 총 변경성 정도는 다음과 같다.

$$TM = TU + \sum_{i=1}^{t} (SUCC(Ci) / 2) \qquad (5)$$

여기서 t는 상속 구조에서 총 클래스의 수이다

클래스 상속 구조의 평균적인 변경성 척도는 다음과 같다.

$$AM = AU + \sum_{i=1}^{t} ((SUCC(Ci) / 2)) / t \qquad (6)$$

예를 들어 그림 2는 AM=35/8=4.38인데 이는 클래스 상속 구조의 변경성의 정도를 나타낸다.

그림 2에 있어서 CK의 DIT는 3인데, Henderson-Sellers[6]는 하나의 클래스 상속 구조에 있어서 DIT가 최고 7을 넘지 않아야 한다고 권장하고 있다. Coad[5]는 각 클래스는 7개 이상의 공용 함수를 갖지 말도록 권장하고 있는데, 이 7이라는 숫자는 인간이 병렬적으로 작업을 수행할 수 있는 최고의 값이기 때문이다. 이와 같은 경험법칙에 의해 본 연구에서 제안한 AM 척도도 최고 7을 넘지 않도록 해야 하며, 일반적으로 1과 7의 중앙값인 4를 권장한다.

4 평가

본 연구에서 제안한 척도를 CK의 DIT와 Henderson-Sellers의 AIT와 비교 평가한다.

4.1 CK의 DIT

CK[4]는 클래스 상속 구조에 있어서 서브클래스의 수를 나타내는 NOC가 커지면 좋지못한 설계가 되므로 상속 계층의 너비보다는 깊이가 좋다고 제안하고 있다. 그러나 유지보수성 척도의 관점에서 보면 타당하다고 볼 수 없다. 예를 들어 그림 3에서 CK는 (b)보다 (a)를 선호한다. 그러나 경험적 관점에서 보면 (b)가 (a)보다 훨씬 이해하기가 쉽고 변경하기도 용이하다

이 두 상속 구조의 이해성과 변경성을 본 연구에서 제안한 AU 척도로 계산하면 다음과 같다.

그림 3(a)의 AU: (1+2+3+4)/4 = 10/4 = 2.5

AM: AU + (3/2+2/2+1/2)/4 = 2.5+3/4 = 3.25

그림 3(b)의 AU: (1+2+2+2)/4 = 7/4 = 1.75

AM: AU + (1/2+1/2+1/2)/4 = 1.75+3/8 = 2.125

즉, 그림 3(b)의 이해성과 변경성이 그림 3(a)보다 좋은데, 이는 경험적으로 생각하는 바와도 같다.

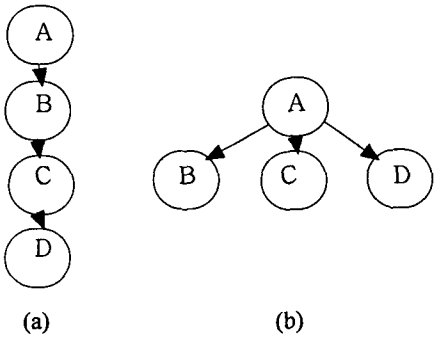

그림 3. 두 개의 클래스 상속 구조 (I)

4.2 Henderson-Sellers의 AID

Henderson-Sellers[6]는 클래스 상속 구조에서 AID(average inheritance depth)를 평균 상속 깊이로 정의한다.

AID of a class inheritance DAG = ∑(depth of each class) / number of classes

예를 들어 그림 4의 클래스 상속 구조를 보자.

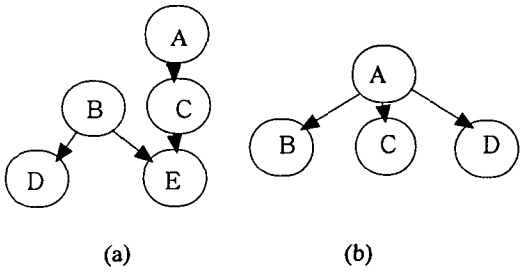

그림 4. 두 개의 클래스 상속 구조 (II)

이를 AID 척도로 계산하면 다음과 같다.

그림 4(a)의 AID: (0+0+1+1+(1+2)/2) / 5 = 0.7

그림 4(b)의 AID: (0+1+1+1) / 4 = 0.75

경험적으로 볼 때 그림 4(b)가 그림 4(a)보다 이해

하기 쉬운데, AID는 반대로 계산이 되었다. 본 연구에서 제안한 AU는 다음과 같이 계산된다.

그림 4(a)의 AU: (1+1+2+2+4) / 5 = 2

그림 4(b)의 AU: (1+2+2+2) / 4 = 1.75

그림 4(b)의 이해성이 그림 4(a)보다 좋게 계산이 되었는데, 이는 경험적으로 생각하는 바와도 같다.

5 결 론

본 논문은 CK의 연구를 바탕으로 이를 확장하여 클래스 상속 구조의 유지보수성을 이해성과 변경성 측면에서 측정할 수 있도록 했다. 그리고 클래스 상속 구조의 두 가지 예를 들어 비교 평가를 함으로써 CK의 척도 및 Henderson-Sellers의 척도보다 우수함을 보였다. 그런데 이 척도들이 더욱 신뢰성을 가지려면 실험적인 검증을 해야 하는데 이는 많은 사례 연구가 요구되는 바, 앞으로의 계속적인 연구 대상이다.

참고 문헌

[1] F. Abreu, "The MOOD Metrics Set," *Proc. ECOOP'95 Workshop on Metrics*, 1995

[2] L. Briand and S. Morasca, "Defining and Validating Measures for Object-Based High-Level Design," *IEEE Transactions on Software Engineering*, Vol.25, No.5, 1999

[3] S. Chidamber and C. Kemerer, "A Metrics Suite for Object Oriented Design," *IEEE Transactions on Software Engineering*, Vol.20, No.6, pp. 476-493, 1994

[4] S. Chidamber and C. Kemerer, ""Towards a metric suite for object-oriented design," *Proc. OOPSLA'91, Sigplan Notices*, 26(11), pp. 197-211, 1991

[5] P. Coad, "OOD Criteria, Part3," Journal of Object Oriented Programming, pp.67-70, Sep. 1991

[6] B. Henderson-Sellers, *Object-Oriented metrics: measures of complexity*, Prentice Hall, 1996

[7] E. Horowitz and S. Sahni, *Fundamentals of Data Structures in C*, Computer Science Press, 1993

[8] T. McCabe, L. Dreyer, A. Dunn and A. Waston, "Testing an object-oriented application," Quality Assurance Institute, pp. 21-27, 1994.

[9] L. Tahvildari and A. Singh, "Categorization of Object-Oriented Software Metrics," *0-7803-5957-7/00/, IEEE,* pp. 235-239, 2000

[10] D. Tegarden, S. Sheetz, and D. Monarchi, "A software complexity model of object-oriented systems," *Decision Support Systems* 13(34), pp. 241-262, 1992