한국 퍼지 및 지능시스템 학회 2002, Vol. 12, No. 6, pp. 537-542

클래스 상속 구조의 유지보수성에 관한 척도와 평가

Metrics for Maintainability of Class Inheritance Structures and it's Evaluation

정 홍

Hong Chung

계명대학교 공학부 컴퓨터공학과

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 Chidamber와 Kemerer의 DIT(Depth of Inheritance Tree)와 NOC(Number of Children) 척도에 대한 실험적 관계를 중심으로 Gursaran이 실험한 결과에 적용하여 본 척도의 타당성을 보였다.

ABSTRACT

We propose new metrics for understandability and modifiability of class inheritance structures based on the object-oriented metrics suggested by Chidamber and Kemerer. The metrics are evaluated using the results of Gursaran's experiments which validated the empirical relation of DIT(Depth of Inheritance Tree) and NOC(Number of Children) metrics of Chidamber and Kemerer.

Key Words : object-oriented software metrics, class inheritance hierarchy, 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 수년간 많은 학자들이 객체지향 소프트웨어를 위한 별도의 척도가 필요한지, 필요하다면 무엇이 포함되어야 하는지를 논의해 왔다[12]. 초기에는 절차중심 프로그래밍에 사용되는 고전적 척도를 확장하는데 주력했으나[10,13], 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 대상 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. 그리고 CK의 척도에 대한 실험적 관계(empirical relation) 즉, 관점(viewpoint)을 중심으로 Gursaran[6]이 실험한 결과에 적용하여 본 논문에서 제안하는 클래스 상속구조의 유지보수성 척도의 타당성을 보이고자 한다.

2절에서는 본 논문의 기초가 되는 CK[3,4]의 DIT와 NOC를 기술하고 그의 문제점을 분석하며, 3절에서는 문제점을 개선하고 이를 기반으로 한 클래스 상속 구조의 유지보수성 척도를 제안한다. 그리고 4절에서는 Gursaran[6]의 실험 결과를 적용하여 본 논문에서 제안한 척도의 타당성을 평가하고, 5절에서는 결론과 앞으로의 연구방향을 제시한다.

접수일자 : 2002년 5월 15일
완료일자 : 2002년 9월 5일

2. CK의 DIT와 NOC

본 논문에서 제안하고자 하는 클래스 상속 구조의 유지보수성에 대한 척도는 CK[3,4]의 DIT와 NOC 척도를 기초로

하고 있다. 따라서 본 절에서는 DIT와 NOC 척도에 대하여 기술하고 이의 문제점을 분석하며, 본 논문에서 제안하고자 하는 척도에 사용할 수 있도록 개선하고자 한다.

2.1 DIT(Depth of Inheritance Tree)

클래스의 DIT 척도는 클래스의 상속 깊이로서, 다중상속인 경우 노드로부터 루트까지의 최대 깊이이다. 어느 클래스의 DIT는 얼마나 많은 조상 클래스로부터 영향을 받는가를 측정한다. CK는 다음과 같은 세가지 관점을 기술하고 있다.

1) 클래스 계층에서 클래스가 깊어질수록 메소드의 상속도가 높아져 메소드의 행위를 예측하기가 복잡해진다.
2) 깊은 트리는 더 많은 메소드와 클래스가 관련되므로 설계 복잡도가 커진다.
3) 클래스가 깊이 있을수록 상속 메소드의 재사용 가능성은 커진다.

예를 들어 그림 1의 클래스 상속 구조에서 클래스 "대학구성원"은 루트 클래스이므로 DIT(대학구성원)=0이며, 클래스 "학생"과 "교수"는 루트로부터 거리가 각각 1이므로 DIT(학생)=DIT(교수)=1이다. 그리고 클래스 "학부생"과 "대학원생"은 루트로부터 거리가 각각 2이므로 DIT(학부생)=DIT(대학원생)=2이다.

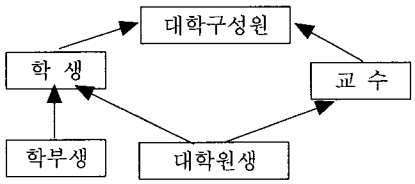


그림 1. 클래스 상속 구조
Fig. 1. A class inheritance structure

Li[9]는 DIT 척도의 정의와 이론적 근거에 모순성이 있다고 지적하고 있다. 이론적 근거에서 "DIT는 얼마나 많은 조상 클래스로부터 영향을 받는가를 측정한다"라고 하고 있는데, 이는 DIT가 조상 클래스의 수를 측정해야 함을 나타내는 것이다. 그러나 DIT의 정의는 "상속 트리에서 경로의 길이" 즉, 두 노드간의 거리를 측정한다라고 되어 있는데, 이는 이론적 근거에서 제시한 측정 방법과 맞지 않다. 예를 들어 그림 1에서 정의에 의하면 클래스 "학부생"과 "대학원생"은 루트까지의 거리가 다같이 2이다. 그러나 이론적 근거에 의하면 각각 2와 3으로 달라야 한다. 객체지향 설계에서 클래스의 구조를 이해해야 하는 이해성 측면에서 볼 때 클래스 "대학원생"을 이해하려면 "학생"과 "교수"를 이해해야 하고, "학생"이나 "교수"를 이해하려면 "대학구성원"도 이해해야 하는 등 3개 클래스 모두를 이해해야 한다. 따라서 본 논문에서는 DIT의 정의를 "조상 클래스의 수"로 재정의 하여 사용하고자 한다. "조상 클래스의 수"는 클래스 상속 구조를 그래프라 볼 때 "노드의 선행자 수"로 볼 수 있다.

2.2 NOC(Number of Children)

NOC는 클래스 계층에서 클래스에 직접 종속된 서브클래스의 수이다. 이는 얼마나 많은 서브클래스가 부모 클래스의 메소드를 상속받는가를 측정하는 척도인데 CK는 다음과 같은 세가지 관점을 기술하고 있다.

1) 상속은 재사용의 한 형태이므로 자식의 수가 많을수록 재사용이 커진다.
2) 자식의 수가 많을수록 부모 클래스의 추상성이 부적합할 가능성이 커진다.
3) 자식의 수는 설계에서 클래스가 미치는 영향을 의미한다.

예를 들어 그림 1에서 클래스 "대학구성원"과 "학생"의 자식이 각각 2이므로 NOC(대학구성원)=NOC(학생)=2이며, "교수"는 1이므로 NOC(교수)=1이다. 그리고 클래스 "학부생"과 "대학원생"은 자식이 없으므로 NOC(학부생)=NOC(대학원생)=0이다.

Li[9]는 또한 NOC도 모호성을 가지고 있다고 지적하고 있다. 즉, 정의에서 NOC 척도는 상속에 의하여 영향을 직접 받는 범위를 측정한다고 하고 있다. 그런데 영향을 받는다는 것은 직접이든 간접이든 다같이 받는 것인데, 직접 상속받는 자식 클래스의 수만 계산한다는 것은 논리상 맞지 않다. 객체지향 설계에서 클래스의 구조를 변경해야 하는 변경성 측면에서 볼 때, 그림 1에서 클래스 "대학구성원"을 변경하면 "학생"과 "교수"도 변경해야 할 가능성이 있고, "학생"을 변경하면 "학부생"과 "대학원생"도 변경해야 할 가능성이 있으며, "교수"를 변경하면 "대학원생"도 변경해야 할 가능성이 있다. 즉, 클래스 "대학구성원"을 변경하면 자손 클래스 모두 변경해야 할 가능성이 있다. 따라서 본 논문에서는 NOC의 정의를 "자손 클래스의 수"로 재정의 하여 사용하고자 한다. "자손 클래스의 수"는 클래스 상속 구조를 그래프라 볼 때 "노드의 후속자 수"로 볼 수 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 그래프(편의상 상속 방향을 거꾸로 표시)에서 임의의 정점 i로부터 임의의 정점 j로의 경로가 있으면 정점 i는 정점 j의 선행자이고, 정점 j는 정점 i의 후속자이다[8]. 따라서 척도 PRED와 SUCC를 다음과 같이 정의한다.

PRED(k) : 정점 k의 선행자의 총 수
SUCC(k) : 정점 k의 후속자의 총 수

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

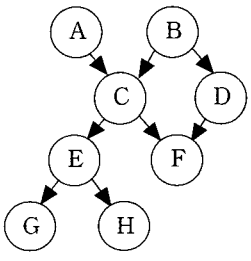


그림 2. 클래스 상속 그래프
Fig. 2. A class inheritance graph

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 것

이며, 변경 활동에 있어서 시스템 변경성은 프로그램 구조나 클래스 상속 구조를 어느 정도 쉽게 변경할 수 있는가 하는 것이다. 따라서 본 연구에서는 클래스 상속 계층의 유지보수성을 이해성 척도와 변경성 척도로 분리하여 제안한다.

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 클래스 F를 이해하려면 F 자체뿐만 아니라 상속을 하는 슈퍼클래스인 C와 D도 이해해야 한다. 또한 클래스 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 상속 그래프의 특정 클래스 Ci의 이해성 척도 U를 다음과 같이 정의한다.

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

그리고 클래스 상속 구조의 총 이해성 정도 TU는 다음과 같다.

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

여기서 t는 상속 구조의 총 클래스의 수이다.

프로그램의 이해도나 복잡도를 거론할 때 규모가 큰 프로그램이 작은 것보다 복잡하다고 하는 것이 일반적이다. 그러나 두 개 이상의 프로그램간의 상대적인 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[7]. 따라서 클래스 상속 구조의 평균 이해성 척도는 다음과 같이 표현할 수 있다.

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

그림 2의 예를 들면 AU=26/8=3.25인데 이 값은 이 클래스 상속 구조의 이해성 정도를 나타내는 것이다.

3.2 변경성 척도

그림 2에서 클래스 C의 내용을 변경하려면 먼저 그것을 이해해야 하는데, C의 이해도는 3.1절에 기술한 U(C)이다. 만약 클래스 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 서브클래스들 중 평균적으로 반은 변경된다고 볼 수 있으므로 클래스 C의 변경성 정도는 U(C)+SUCC(C)/2라고 할 수 있다. 따라서 상속 그래프의 특정 클래스 Ci의 변경성 척도 M은 다음과 같이 정의한다.

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

그리고 클래스 상속 구조의 총 변경성 정도는 다음과 같다.

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

여기서 t는 상속 구조에서 총 클래스의 수이다

따라서 클래스 상속 구조의 평균적인 변경성 척도는 다음과 같이 표현할 수 있다.

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

예를 들어 그림 2는 AM=3.25+9/8=4.38인데 이는 클래스 상속 구조의 변경성의 정도를 나타내는 것이다.

그림 2에 있어서 CK[3,4]의 DIT를 계산해보면 가장 큰 값이 3인데, Henderson-Sellers[7]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 경험 법칙에 의해 본 연구에서 제안한 AM 척도도 최고 7을 넘지 않도록 해야 하며, 일반적으로 1과 7의 중앙값인 4를 권장하고자 한다.

4 평가

본 연구에서 제안한 척도 중 이해성 척도는 PRED를 기반으로 하고 있으며, 변경성 척도는 SUCC를 기반으로 하고 있다. PRED로부터 유도한 이해성 척도 AU와 PRED 및 SUCC로부터 유도한 AM은 프로그램의 이해 과정이나 변경 과정에 따른 직관적이고도 경험적 방법으로 유도한 것이다. 따라서 본 논문에서 제안한 척도에서 사용되는 PRED와 SUCC를 CK[4]의 DIT와 NOC를 비교하여 타당성을 검토하고자 한다. 타당성의 검토 작업은 DIT와 NOC 척도에 대한 실험적 관계 즉, 관점을 중심으로 Gursaran[6]이 실험한 결과에 적용하여 본 척도와의 비교를 하고자 한다.

Gursaran은 CK의 DIT와 NOC의 관점을 그대로 사용 사용하는 대신 "클래스가 조상의 특성에 의해 영향을 받는 범위"와 "클래스가 자손에 미칠 영향의 가능성"이라는 관점으로 일반화 시켰다. 그 이유는 "상속의 깊이"와 "직접 자식의 수"라는 용어로 제한하는 것보다는 클래스의 이해성과 변경성을 결정하는데 있어서 더 합당하기 때문이다.

4.1 검증을 위한 상속 계층

검증을 위해 사용될 상속 계층은 DIT를 위한 관점 표현을 검증하기 위해 그림 3에 5개의 계층 구조를 표시하며, NOC를 위한 관점 표현을 검증하기 위해 그림 4에 5개의 계층 구조를 표시했다. 각 그림에 있어서 * 표시가 있는 클래스에 대해 각 관점의 등급을 계산한다. 등급이란 * 표시가 있는 클래스를 이해하거나 변경하는데 있어서의 어려움의 상대적 정도를 나타낸다.

그림 3에서 DIT를 위한 계층의 5가지의 변형을 표현하고 있는데, 이는 다음과 같은 특징을 비교하고자 하는 것이다.

(1) *표가 붙어있는 클래스의 깊이를 변경: 계층 A와 C, 계층 B와 D
(2) *표가 붙어있는 클래스의 단순 상속과 다중 상속: 계층 A와 B, 계층 C와 D
(3) 조상에 있어서의 단순 상속과 다중 상속: 계층 C와 E
(4) *표가 붙어있는 클래스간의 깊이, 상속의 성격, 조상의 수의 차이: 계층 A와 E

이들 계층을 선택한 것은 깊이가 고정되든 변하든, 계층의

성격에 작든 크든 차이가 존재할 때 DIT가 관점을 정확하게 표현하고 있는지를 실험하고자 하는 것이다.

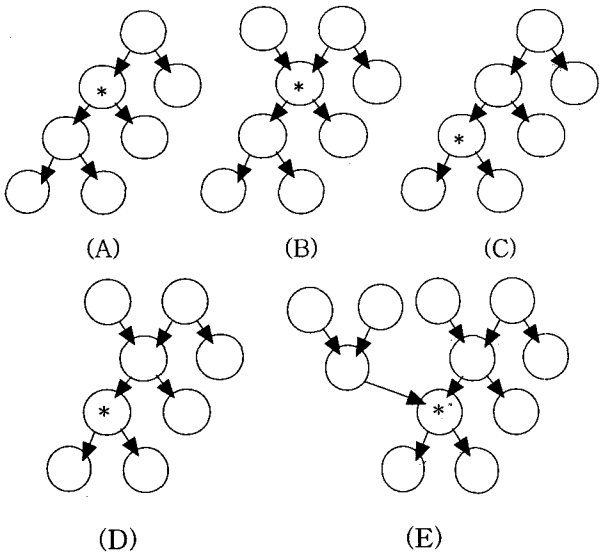


그림 3. DIT 검증을 위한 클래스 상속 계층
Fig. 3. A class inheritance hierarchies for DIT validation

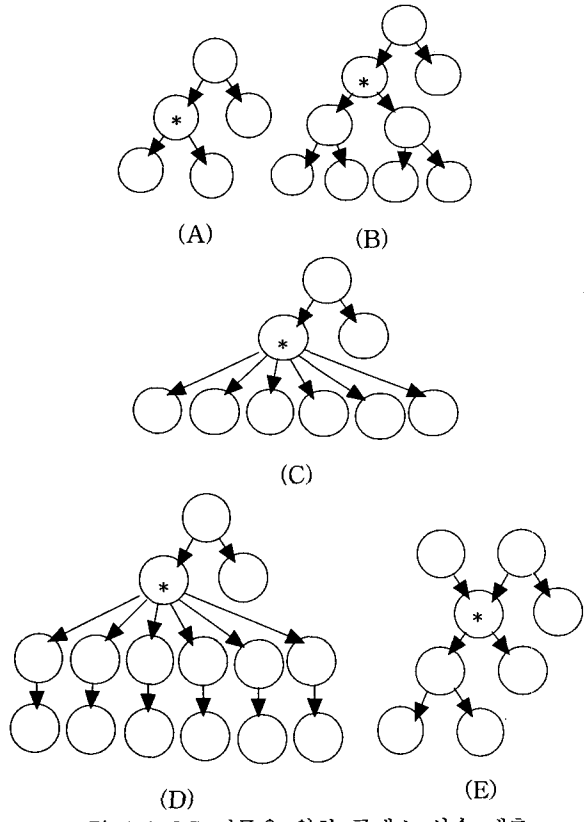


그림 4. NOC 검증을 위한 클래스 상속 계층
Fig. 4. A class inheritance hierarchies for NOC validation

그림 4에서 NOC를 위한 계층의 5가지의 변형을 표현하고 있는데, 다음과 같은 특징을 비교하고자 한다.

(1) *표가 붙어있는 클래스간에 직접 자식의 수의 차이: 계층 A와 C
(2) *표가 붙어있는 클래스간에 직접 혹은 간접 자식의 수의 차이: 계층 A와 B
(3) 한 클래스의 직접자식의 수와 다른 클래스의 동일한 수의 자손의 수: 계층 B와 C
(4) *표가 붙어있는 클래스간에 많은 직접 자식의 수를 가진 클래스와 많은 자손의 수를 가진 클래스: 계층 C와 D
(5) 다중 상속을 가진 클래스와 단순 상속을 가진 클래스: 계층 E와 나머지

이들 계층을 선택한 것은 깊이가 고정되든 변하든, 계층의 성격에 작든 크든 차이가 존재할 때 NOC가 관점을 정확하게 표현하고 있는지를 실험하고자 하는 것이다.

4.2 실험 방법 및 결과

Gursaran[6]은 그림 3과 그림 4에 대해 *표가 붙어 있는 클래스들에 대해 각 관점별 등급을 구하기 위해 다음 예와 같은 질문지를 관점별로 만들었다.

• DIT 관점 1에 대한 질문
1) 계층 A에 있는 *표 클래스는 계층 B에 있는 *표 클래스보다 더 많은 멤버함수를 상속 받으므로 ( )만큼 더 행위를 예측하기가 더 복잡하다.
2) 계층 B에 있는 *표 클래스는 계층 A에 있는 *표 클래스보다 더 많은 멤버함수를 상속 받으므로 ( )만큼 더 행위를 예측하기가 더 복잡하다.

• DIT 관점 2에 대한 질문
1) 계층 A에 있는 *표 클래스는 계층 C에 있는 *표 클래스보다 더 많은 멤버함수와 클래스를 포함하므로 ( )만큼 더 설계가 더 복잡하다.
2) 계층 C에 있는 *표 클래스는 계층 A에 있는 *표 클래스보다 더 많은 멤버함수와 클래스를 포함하므로 ( )만큼 더 설계가 더 복잡하다.

• DIT 관점 3에 대한 질문
1) 계층 A에 있는 *표 클래스는 계층 C에 있는 *표 클래스보다 ( )만큼 더 상속 클래스의 멤버 함수의 재사용 가능성이 크다.
2) 계층 C에 있는 *표 클래스는 계층 A에 있는 *표 클래스보다 ( )만큼 더 상속 클래스의 멤버 함수의 재사용 가능성이 크다.

• NOC 관점 1에 대한 질문
1) 계층 A에 있는 *표 클래스는 계층 B에 있는 *표 클래스보다 상속을 통하여 ( )만큼 더 많은 재사용을 보인다.
2) 계층 B에 있는 *표 클래스는 계층 A에 있는 *표 클래스보다 상속을 통하여 ( )만큼 더 많은 재사용을 보인다.

• NOC 관점 2에 대한 질문
1) 계층 A에 있는 *표 클래스는 계층 C에 있는 *표 클래스보다 ( )만큼 더 서브클래스화가 잘못 되어 있다.
2) 계층 C에 있는 *표 클래스는 계층 A에 있는 *표 클래스보다 ( )만큼 더 서브클래스화가 잘못 되어 있다.

• NOC 관점 3에 대한 질문
1) 계층 A에 있는 *표 클래스는 계층 E에 있는 *표 클래스보다 설계에 영향을 많이 줌으로 ( )만큼 더 많은 테스트를 해야 한다.
2) 계층 E에 있는 *표 클래스는 계층 A에 있는 *표 클래스보다 설계에 영향을 많이 줌으로 ( )만큼 더 많은 테스트를 해야 한다.

실험을 위해 소프트웨어공학, C++에 의한 객체지향 프로그래밍 등을 수강한 평균 이상의 성적을 취득한 13명의 대학원생들에게 조사 내용과 질문에 대하여 설명하고, 이들이 독립적으로 조사 작업에 응하고 질문에 답하도록 했다. 질문에

대한 답은 각자 주관적인 판단으로 0~1의 상대적인 값을 매기도록 했다. 그리고 질문에 대한 AHP(Analytic Hierarchy Process) 분석[11]을 수행할 때 응답에 대한 이유를 학생들과 함께 토의를 통하여 분석하였다. 분석 결과는 표 1과 같다. 수치는 각 응답자에 대한 priority vector 값의 평균이다. NOC 관점 2에 대해서는 응답이 없는 것이 많아 제외하였다.

표 1. 그림 3과 그림 4의 관점별 priority vector
Table 1. Priority vector of figure 3 and 4 by viewpoints

		A	B	C	D	E
그림 3: DIT	관점 1	0.03	0.07	0.10	0.19	0.59
	관점 2	0.04	0.09	0.07	0.20	0.59
	관점 3	0.04	0.10	0.08	0.22	0.54
그림 4: NOC	관점 1	0.07	0.11	0.32	0.39	0.09
	관점 3	0.04	0.15	0.18	0.53	0.07

이를 등급 순으로 표시하면 표 2와 같다. 표 2에는 평균이 아닌 다수결의 의견일치에 의한 등급도 표시했다. 여기서 괄호 표시는 등급 구분이 어려움을 나타낸다.

표 2. 그림 3과 그림 4의 관점별 등급
Table 2. Rating of figure 3 and 4 by viewpoints

		등급(평균)	등급(의견일치)
그림 3: DIT	관점 1	E-D-C-B-A	E-D-(C,B)-A
	관점 2	E-D-B-C-A	E-D-(C,B)-A
	관점 3	E-D-B-C-A	E-D-(C,B)-A
그림 4: NOC	관점 1	D-C-B-E-A	D-C-(B,E)-A
	관점 2		(D,C)-(B,E,A)
	관점 3	D-C-B-E-A	D-(C,B)-E-A

그림 3과 그림 4에 대한 CK의 DIT와 NOC 그리고 본 논문의 PRED와 SUCC를 계산하면 표 3과 같다.

표 3. 그림 3과 그림 4에 대한 척도의 계산
Table 3. Computation of metrics for figure 3 and 4

	A	B	C	D	E
그림 3 : DIT	2	2	3	3	3
PRED	1	2	2	3	6
그림 4 : NOC	2	2	6	6	2
SUCC	2	6	6	12	4

표 2의 관점별 등급을 하나로 종합하고 표 3의 척도를 등급화하여 비교표를 만들면 표 4와 같이 된다.

표 4. 실험치와 척도의 등급 비교
Table 4. Comparison of ranking between value of experiment and metrics

	실험치	DIT	NOC	PRED	SUCC
그림3	E-D-(B,C)-A	(E,D,C)-(B,A)		E-D-(B,C)-A	
그림4	D-C-B-E-A		(D,C)-(B,E,A)		D-(C,B)-E-A

표 4를 보면, 그림 3에 대해 실험치와 비교해 볼 때 DIT는 차이가 많이 나나 PRED는 일치하며, 그림 4에 대해 실험치와 비교해 볼 때 NOC는 차이가 나나 SUCC는 거의 차이가 없다. 이상과 같이 본 논문의 PRED 및 SUCC는 CK의 DIT 및 NOC보다 우수하며 또한 실험치와도 거의 일치함을 알 수 있다. 따라서 PRED로부터 유도한 이해성 척도 AU와 PRED 및 SUCC로부터 유도한 AM은 시스템 설계 단계에서 시스템의 유지보수성을 판단하는데 효과적인 척도가 될 것으로 판단된다.

5 결 론

본 논문은 객체지향 설계의 척도에 관한 CK의 연구를 바탕으로 이를 확장하여 클래스 상속 구조의 유지보수성을 이해성과 변경성 측면에서 측정할 수 있도록 새로운 척도 AU와 AM을 제안했다. 그리고 이들의 타당성의 평가는 CK의 DIT와 NOC 척도에 대한 관점을 중심으로 Gursaran이 실험한 결과에 적용하여 본 척도의 기반이 되는 PRED와 SUCC가 DIT와 NOC보다 우수함을 보였다. 그런데 본 논문에서 제안한 이해성 척도와 변경성 척도는 PRED와 SUCC를 기반으로 유지보수성을 직관적, 경험적으로 유도한 척도이므로 이의 신뢰성을 더욱 확보하려면 이 척도를 사용한 직접적 실험을 해야 하는데 이는 앞으로의 계속적인 연구 대상이다.

참 고 문 헌

[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] R. Gursaran, "Viewpoint representation validation: a case study on two metrics from the Chidamber and Kemerer suite," The Journal of Systems and Software 59(1), pp.83-97, 2001

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

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

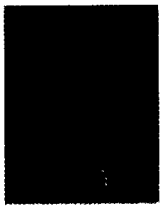
[9] W. Li, "Another metric suite for object-oriented programming," The Journal of Systems and Software, Vol.44, pp. 155-162, 1998

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

[11] T. Saaty, The Analytic Hierarchy Process, RWS Publications, Pittsburg, PA, 1990

[12] L. Tahvildari and A. Singh, "Categorization of Object-Oriented Software Metrics," 0-7803-5957-7/00/, IEEE, pp. 235-239, 2000
[13] D. Tegarden, S. Sheetz, and D. Monarchi, "A software complexity model of object-oriented systems," Decision Support Systems 13(34), pp. 241-262, 1992

저 자 소 개



정 홍(Hong Chung)
1972년 : 한양대학교 원자력공학과 (공학사)
1976년 : 고려대학교 경영대학원 생산관리 (경영학석사)
1996년 : 대구가톨릭대학교 전산통계학과 (이학석사)
1999년 : 대구가톨릭대학교 전산통계학과 (이학박사)
1972년~1981년 : 한국과학기술연구원 선임연구원
2000년~2001년 : 미국 Washington State University 연구교수
1981년~현재 : 계명대학교 컴퓨터공학과 부교수

관심분야 : 지능정보시스템, 소프트웨어공학
E-mail : jhong@kmu.ac.kr