한국 퍼지 및 지능시스템 학회 2002, Vol. 12, No. 4, pp. 328-333

클래스 상속구조에 대한 경험적 복잡성 척도

A Heuristic Metric for Measuring Complexity of Class Inheritance Structures

정홍 · 김태식

Hong Chung and Taesik Kim

계명대학교 공학부

School of Engineering, Keimyung University

요 약

상속구조의 계층이 깊어질수록 재사용성은 좋으나 이해하기가 어려워지고 유지보수가 힘들어 진다. 반대로 계층이 얕을수록 추상성이 부족하나 이해성과 수정성이 좋아진다. 따라서 시스템의 유지보수성을 위해서는 깊은 상속구조를 분리하여 얕은 상속구조로 만드는 것이 바람직하다. 본 연구에서 제안한 복잡성 척도는 Chidamber and Kemerer의 상속구조에 관한 척도인 DIT(Depth of Inheritance Tree)와 NOC(Number Of Children)를 기초로 Li가 지적한 Chidamber and Kemerer 척도의 모호성을 해결한 통합적 척도로서, 클래스 상속구조의 복잡성 측정에 있어서 조상 클래스 수, 자손 클래스 수, 상속구조의 깊이를 고려한 간단하고 휴리스틱한 척도이다. 이는 상속구조의 분리에 있어서 복잡도를 정량적으로 평가할 수 있는 정보를 제공한다.

Abstract

The deeper the hierarchy of a inheritance structure is, the better the reusability of the structure is, but the more difficult the understandability and the maintainability of it is. On the contrary, the shallower the hierarchy is, the worse the abstraction of the inheritance structure is, but the better the understandability and modifiability of it is. Therefore, it is to be desired that a deep hierarchy of a inheritance structure should be split to be shallow for the maintainability of a system. This paper proposed a complexity metric that is based on DIT and NOC of Chidamber and Kemerer, and solved the ambiguity of the metrics of them, which was pointed out by Li. The metric is a simple and heuristic one for measuring the complexity of class inheritance structures by considering the number of ancestor classes and descendant classes and the depth of inheritance hierarchy. This provides a quantitative information for assessing the complexity of a inheritance structure in splitting it.

Key Words : Software Metrics, Inheritance Hierarchy, Software Complexity

1. 서 론

측정(measurement)은 공학적 접근에 있어서 기본적인 역할을 하는데, 이는 소프트웨어 공학에서도 예외가 아니다[2]. 측정을 위한 척도(metrics)는 생산성과 소프트웨어 품질을 측정하고, 에러를 발견하고, 비용과 일정을 예측하고, 프로젝트를 통제하고 제어하는 등 관리에 있어서 필수적이다. 현재 많이 사용되고 있는 기능점(Function Point) 분석, 소프트웨어 과학, Cyclomatic 복잡도 등과 같은 전통적 척도는 초기 세대의 프로그래밍에 초점을 맞추고 있으며, 객체지향 패러다임(클래스, 상속, 캡슐화, 메시지 전달 등) 측면에서는 잘 적용되지 못하고 있다[14].

10여년 이상 학자들은 별개의 객체지향 소프트웨어 척도가 필요한지, 필요하다면 어떤 특성을 포함해야 하는지를 논의해 왔다[15]. 초기 제안은 절차중심 프로그래밍에 사용되는 척도를 객체지향 시스템에도 적용할 수 있도록 확장하는데 초점을 맞추었으나[13,16], 최근은 객체지형 패러다임의 특성에 맞은 척도를 제안하고 있다[1,4,6,7].

CK(Chidamber and Kemerer)[7]는 여섯 가지의 객체지향 척도를 제안하고 Weyuker의 측정이론[18]에 따라 분석적으로 평가했다. CK의 척도를 검증하기 위해 여러 연구가 수행되었는데, Basili 등[3]은 CK의 척도를 실험적으로 검증한 결과 여섯 가지의 척도 중 다섯 가지는 오류 가능성이 있는 클래스를 예측하기 위한 품질 지표로 유용하다고 했고, Li[12]는 Kitchenham 등[10]이 제안한 척도평가 framework을 사용하여 CK 척도를 이론적으로 검증하여 몇 가지 결함이 있다는 것을 발견하고 이를 개선한 새로운 척도를 제안했다. Balasubramanian[2]도 CK의 접근법에 대한 결함을 지적하고 몇 가지 새로운 척도를 제안했다. Briand 등[4]은 객체지향 설계를 위한 결합도 측정 척도를 제안하고, 이것이 CK의 척도를 보완하는 품질 지표가 된다고 했다. Abreu[1]는 MOOD(Metrics of Object-Oriented Design)라는 객체지향 설계 척도를 제안하고 측정 원리적 관점에서 이론적으로 검증을 했는데, Harrison 등[8]은 MOOD를 실험적

접수일자 : 2001년 10월 26일
완료일자 : 2002년 5월 2일

으로 검증한 결과 이들 척도는 시스템 수준에서 잘 동작하며 클래스 수준에 있어서 CK 척도와 보완적이라 하고 있다.

대부분의 연구에서 클래스 상속구조를 클래스 계층의 깊이와 클래스 수의 측면에서 측정할 필요성을 제기하고 있는데, 이는 클래스의 상속구조가 객체지향 시스템의 설계에 있어서 매우 중요한 부분을 차지하기 때문이다. 클래스는 객체지향 시스템의 기능적 요구사항을 다루는 가장 중요한 부분으로서 시스템 설계시 가장 먼저 설계를 해야한다. 그리고 상속은 객체를 여러 수준의 클래스로 분류할 수 있도록 함으로써 객체지향 설계를 용이하게 한다[17].

본 연구는 클래스 상속구조에 있어서 객체지향 시스템 모델링을 위한 시스템 수준의 복잡도 측정을 위한 새로운 척도를 제안한다. 이는 CK[7]의 상속구조에 관한 척도인 DIT(Depth of Inheritance Tree)와 NOC(Number Of Children)를 기초로 Li[12]가 지적한 CK 척도의 모호성을 해결한 통합적 척도로서, 클래스 상속구조의 복잡성 측정에 있어서 조상 클래스 수, 자손 클래스 수, 상속구조의 깊이를 고려한 간단하고 휴리스틱한 척도이다.

2. 상속구조에 관한 척도

클래스 상속구조와 관련된 기존의 척도와 이를 제안한 연구자는 다음 표 1과 같다.

표 1. 상속구조에 관한 기존의 척도
Table 1. Existing measures for inheritance structures

척 도	의 미	연구자
DIT	상속계층에서 어떤 클래스로부터 루트 클래스까지의 길이	CK[7]
NOC	상속계층에서 어떤 클래스로부터 직접적으로 상속받는 클래스의 수	
AID(Average Inhertiance Depth)	상속계층에서 각 클래스의 평균 깊이	Henderson-Sellers[9]
NAC(Number of Ancestor Class)	상속계층에서 어떤 클래스의 조상 클래스의 수	Li[12]
NDC(Number of Descendent Classes)	상속계층에서 어떤 클래스의 자손 클래스의 수	
CLD(Class-to Leaf Depth)	상속계층에서 어떤 클래스 밑에 있는 최대 깊이의 클래스까지의 거리	Tegarden at al.[16]
NOA(Number Of Ancestor)	상속계층에서 어떤 클래스에 직간접으로 상속을 하는 조상 클래스의 수	
NOP(Number Of Parents)	어떤 클래스에 직접적으로 상속하는 부모 클래스의 수	Lake and Cook[11]
NOD(Number Of Descendents)	어떤 클래스로부터 직간접적으로 상속을 받는 클래스의 수	

이중 가장 많이 거론되고 있는 것이 CK[7]의 DIT와 NOC인데, 다음에 간단히 기술하고 문제점을 분석해 본다.

- DIT

· 정의: DIT는 클래스의 상속계층에 있어서 상속의 깊이이다. 다중상속인 경우 상속 트리의 루트노드로부터 가장 긴 거리를 취한다.
· 이론적 근거 : DIT는 얼마나 많은 조상 클래스가 이 클래스에 영향을 미칠 가능성이 있는가를 측정한다.
· 관점 : 1) 상속계층에서 클래스가 깊어질수록 메소드 상속의 정도도 커지게 되어 메소드의 동작을 예측하기 어렵게 한다. 2) 상속계층이 깊어질수록 더 많은 메소드와 클래스가 포함되어 설계 복잡도가 크게 된다. 3) 특정 클래스가 상속계층에 깊이 있을수록 상속된 메소드의 재사용성은 커진다.
· 예 : 그림 1에서 클래스 D에서 루트 클래스 A까지의 거리가 2이므로 DIT(D)=2이며, 다중상 속을 가진 클래스 E에서 루트 클래스 A까지의 거리는 1과 2중 2가 크므로 DIT(E)=2이다.

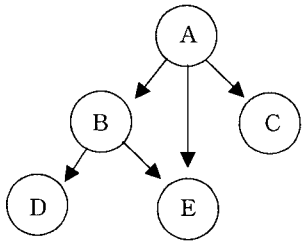

그림 1. 상속 트리
Fig. 1. Inheritance tree

DIT는 어떤 클래스가 조상 클래스의 특성으로부터 영향을 받는 범위로서, 상속계층의 루트 클래스로부터 어떤 클래스까지의 거리를 나타내며, 다중상속 경우는 여러 거리 중 최대 거리를 취한다.

- NOC

· 정의 : NOC는 상속계층에서 직접 종속된 서브클래스의 수이다.
· 이론적 근거 : NOC는 얼마나 많은 서브클래스가 부모 클래스로부터 메소드를 상속받는가를 측정한다.
· 관점 : 1) 상속은 재사용 형태이므로 자식 수가 많을수록 재사용성이 커진다. 2) 자식 수가 많을수록 부모 클래스의 추상성이 부적합할 가능성이 커진다. 3) 자식의 수는 설계에서 하나의 클래스가 얼마나 많은 영향을 미치는가 하는 것을 나타낸다.
· 예 : 그림 1에서 클래스 A는 3개의 서브클래스를 가지므로 NOC(A)=3이며, 클래스 B는 2개의 서브클래스를 가지므로 NOC(B)=2이다.

NOC는 어떤 클래스가 자손 클래스에 미치는 영향을 나타내는 척도로서, 부모 클래스의 메소드를 상속받는 서브클래스의 수이다.

Li[12]는 CK의 DIT에 약간의 모호한 점이 있다고 지적했다. DIT의 이론적 근거에서 "DIT는 얼마나 많은 조상 클래스가 이 클래스에 영향을 미칠 가능성이 있는가

를 측정한다"라고 하고 있는데 이는 어떤 클래스에 직간접으로 상속되는 조상의 수를 의미한다. 그런데 DIT의 정의에서는 상속계층에서 두 노드간의 거리인 경로 길이로 하고 있어서 정의와 이론적 근거가 일치하지 못한다. 이러한 모호성은 다중 루트와 다중 상속을 동시에 가진 경우에 발생하는데, 이는 다중 상속 트리에서는 어떤 클래스로부터 루트 클래스까지 거리가 같지 않는 경우가 발생하기 때문이다. 예를 들어 그림 2에서 클래스 F는 루트 클래스까지의 거리가 1도 있고 2도 있으므로 DIT(F)를 어느 것으로 결정할 지 모호하다.

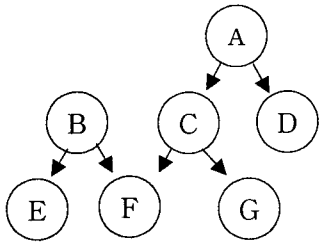

그림 2. 다중 상속 트리
Fig. 2. Multiple inheritance tree

Li는 또 NOC에도 약간의 모호성이 있다고 했다. NOC의 이론적 근거와 관점에서는 NOC가 어떤 클래스가 상속에 의해 영향을 미치는 범위를 측정하는 것이라 하고, 정의에서는 어떤 클래스가 직접 상속하는 서브클래스의 수라고 하고 있다. 실제로 클래스가 영향을 미치는 것은 직접이든 간접이든 모두 포함되어야 하는데, 직접 상속하는 것으로만 제한하는 것은 적합하지 못하다.

3. 경험적 복잡성 척도

클래스 상속구조의 복잡도는 어떤 클래스에 영향을 미치는 모든 조상 클래스와 그 클래스가 영향을 주는 모든 자손 클래스의 상태가 어떤지 파악함으로써 측정할 수 있다. 예를 들어 그림 3에서 클래스 I를 이해하려면 클래스 E와 F를 알아야 하고 클래스 E를 이해하려면 클래스 B 및 루트 클래스 A를 알아야 한다. 마찬가지로 클래스 F를 이해하려면 클래스 C와 A를 이해해야 한다. 즉, 클래스 I 외에 5개 클래스의 구조를 알아야 한다. 다시 말하면 상속을 하는 조상 클래스의 수만큼 더 알아야 한다. 이는 DIT의 이론적 근거에서 기술한 바와 같이 클래스에 영향을 미칠 가능성이 있는 모든 조상 클래스를 의미한다. 또 그림 3에서 클래스 B를 수정하려면 클래스 B가 영향을 미치는 서브클래스 D와 E도 변경이 발생하는지 조사해야 한다. 마찬가지로 클래스 E를 수정하려면 그것의 서브클래스 H와 I도 또한 변경이 발생하는지 파악해야 한다. 그러나 E를 변경해야 할 필요가 없다면 H와 I를 조사할 필요가 없다. 즉 클래스 B 외에 이것이 영향을 미치는 2~4개의 서브클래스를 조사해 보아야 한다. 다시 말하면 상속을 하는 자손 클래스 수의 반만큼 수정을 해야 할지 조사해 보아야 한다. 이는 Li[12]가 지적한 바와 같이 직접이든 간접이든 영향을 미치는 모든 자손 클래스를 의미한다.

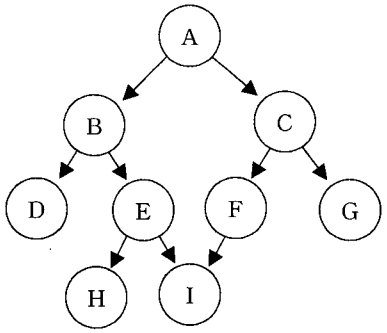

그림 3. 클래스 상속 계층
Fig. 3. Class inheritance hierarchy

Briand[5]의 실험적 연구에 의하면 상속 수준이 깊은 클래스는 얕은 클래스보다 이해하기가 어렵다. 왜냐하면 상속 수준이 깊어질수록 조상 클래스를 일관성 있게 확장 또는 특수화하기 어렵기 때문이다. 따라서 본 연구에서는 상속구조의 깊이를 복잡도의 가중치로 사용하고자 한다.

어떤 클래스 i의 조상 클래스 수를 Pi, 자손 클래스 수를 Ci라 할 때, 클래스 상속구조의 복잡도 CIS(Complexity of Inheritance Structure)는 다음과 같이 경험적으로 정의를 할 수 있다.

$$CIS = (\sum_{i=1}^{t}(1+Pi+Ci/2)) \cdot D \quad (1)$$

여기서 t는 상속구조에서의 클래스 수를 말하며, 1은 자신의 클래스를 의미하고, D는 상속구조의 깊이이다.

객체지향 시스템의 모델에서 시스템이 f개의 상속구조를 가진다면 시스템 전체의 상속구조 복잡도 TCIS(Total Complexity of Inheritance Structure)는 다음과 같다.

$$TCIS = \sum_{j=1}^{f} CISj \quad (2)$$

CIS는 상속구조의 복잡도를 CK처럼 DIT와 NOC라는 별개의 이원적 개념으로 보는 것이 아니라 하나의 통합적 척도로 복잡도를 계산한다.

그림 3의 예에 대한 복잡도를 계산하면 다음과 같다.

A : 1+0+8/2 = 5
B : 1+1+4/2 = 4
C : 1+1+3/2 = 3.5
D : 1+2+0/2 = 3
E : 1+2+2/2 = 4
F : 1+2+1/2 = 3.5
G : 1+2+0/2 = 3
H : 1+3+0/2 = 4
I : 1+5+0/2 = 6

CIS = (5+4+3.5+3+4+3.5+3+4+6) · 4 = 144

즉, 상속구조의 복잡도는 144이다.

이상과 같이 상속구조의 복잡도를 측정하는데 있어서 경험적 절차를 정리하면 다음과 같다.

① 각 클래스의 조상 클래스 수와 자손 클래스 수를 계산한다.
② 조상 클래스와 자손 클래스 수를 합하고 상속구조의 깊이를 가중치로 곱한다.
③ 여러 상속구조의 복잡도를 합한다.

여기서 정의한 상속구조의 복잡도는 각 클래스의 내부 복잡도를 균일하다고 가정하고 정의한 것이다. 따라서 이 정의는 시스템 개발 수명주기에서 시스템 모델링 단계의 상속구조 설계시 복잡도를 측정하고자 하는 것이다. 이를 시스템 구현 단계로 확장하여 복잡도를 측정하려면 클래스 내부구조의 복잡도를 계산하여 이 값을 각 클래스의 가중치로 사용하면 된다. 즉, 클래스는 속성과 메소드로 구성되므로 클래스 내부구조의 복잡도 CISC(Complexity of Internal Structure of Class)는 다음과 같이 정의된다.

$$CISC = a \cdot \text{속성의 복잡도} + b \cdot \text{메소드의 복잡도} \quad (3)$$

여기서 속성의 복잡도는 속성의 구조(변수, 배열, 구조 등)에 따른 가중치의 합이고, 메소드의 복잡도는 각 메소드에 대한 Cyclomatic 수의 합이다. a와 b는 속성과 메소드의 상대적 복잡도 상수로서 앞으로의 연구 대상이다. 구현 단계에서의 상속구조의 복잡도에 대한 사항은 본 연구의 목적이 아니므로 더 이상 언급하지 않는다.

4. 비교 및 평가

본 논문에서 제안한 복잡성 척도를 CK가 제안한 DIT와 NOC와 비교해 보기 위해 그림-4와 같은 상속구조의 복잡도를 계산해 본다.

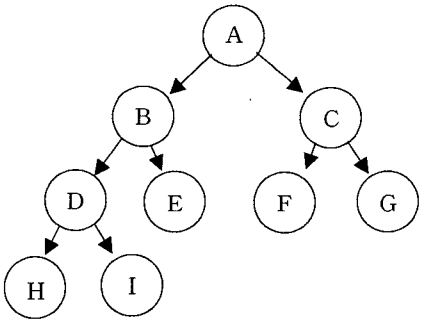

그림 4. 상속 구조
Fig. 4. Inheritance structure

CIS : ((1+8/2)+(1+1+4/2)+(1+1+2/2)+(1+2+2/2)+(1+2)+(1+2)+(1+2)+(1+3)+(1+3))×4 = 132
CK의 DIT = 0+1+1+2+2+2+2+3+3 = 16
CK의 NOC = 2+2+2+2+0+0+0+0+0 = 8

잘 설계된 객체지향 시스템은 하나의 큰 상속구조를 가지는 것보다 작은 상속구조의 모임(forest of classes)을 갖도록 하는 것이 좋다[3,14]. 따라서 그림-4의 복잡도가 너무 크다고 가정할 때 그림-5와 같이 루트를 분할한 2개의 상속구조로 분리하고 복잡도를 계산해 본다.

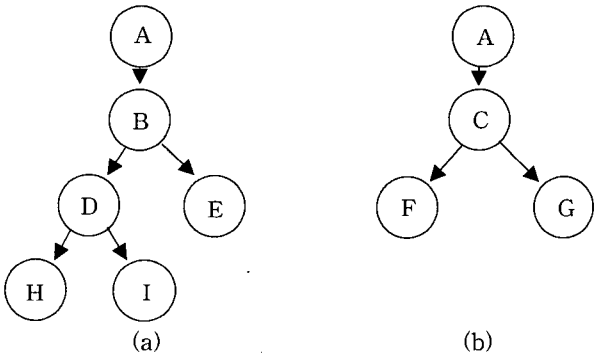

그림 5. 분리된 상속 구조
Fig. 5. Decomposed inheritance structure

(a)의 CIS : ((1+5/2)+(1+1+4/2)+(1+2+2/2)+(1+2)+(1+3)+(1+3))×4=90
(b)의 CIS: ((1+3/2)+(1+1+2/2)+(1+2)+(1+2))×3 = 34.5
TCIS = 90+34.5=124.5
CK의 DIT (a) 0+1+2+2+3+3=11
(b) 0+1+2+2=5
합계=11+5=16
CK의 NOC (a) 1+2+2+0+0+0=5
(b) 1+2+0+0=3
합계 : 5+3=8

상속구조를 분리함으로써 클래스의 재사용성은 나빠지나 CIS 척도로 계산해본 복잡도는 줄어든다. 그러나 재사용성이 줄어듦에도 불구하고 CK의 척도에는 변화가 없다. 이는 CK의 척도는 상속구조의 복잡도 측정에는 적합성이 부족하다는 것을 알 수 있다.

그런데 본 척도는 상속구조가 포화트리(full tree)에 가까운 특수한 경우에는 적용되지 못한다. 왜냐하면 이와 같은 상속구조는 분리해도 트리의 깊이가 줄어들지 않기 때문이다.

CK[6]는 클래스 상속구조에 있어서 폭이 넓은 것보다는 깊이가 깊은 것이 좋다고 하고 있다. 그러나 복잡도 측면에서 볼 때는 그렇지 않다. 예를 들어 그림 6을 보자.

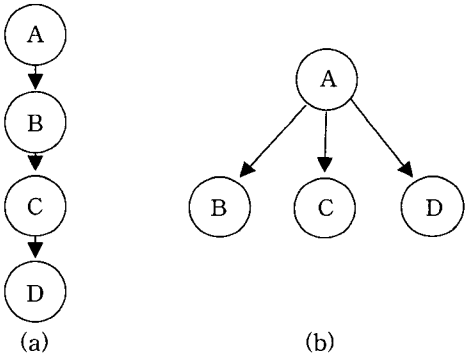

그림 6. 깊이가 깊은 상속구조와 폭이 넓은 상속구조
Fig. 6. Deep structure and wide structure of inheritance

(a)의 CIS: ((1+3/2)+(1+1+2/2)+(1+2+1/2)+(1+3))×4=52
(b)의 CIS: ((1+3/2)+(1+1)+(1+1)+(1+1))×2=17

이와 같이 그림 6에서 (b)가 (a)보다 복잡도가 매우 낮은 것으로 계산이 되었으며, 이는 우리들이 경험적으로 생각하는 바와도 같다.

5. 결 론

객체지향 소프트웨어의 모든 특성을 측정할 수 있는 하나의 척도는 존재하지 않는다. 특히 복잡도를 측정하는 좋은 접근은 필요한 특성을 분리하여 그에 적합한 척도를 만드는 것이다. 객체지향 소프트웨어의 여러 가지 특성중 상속구조의 복잡성은 클래스 상호간의 관련성에 가장 영향을 많이 받는다. 따라서 본 연구에서는 상속구조의 복잡성에 관한 척도를 조상 클래스의 수, 자손 클래스의 수, 상속구조의 깊이를 기반으로 하여 경험적이고도 실제적인 방법으로 접근했다. 즉, 상속구조의 클래스에 대한 조상 클래스의 수와 자손 클래스의 수를 합하여 각 클래스의 복잡도를 구하고, 전체 클래스를 합하여 상속구조의 깊이를 가중치로 곱하는 간단하고 이해하기 쉬운 계산 방법이다.

상속구조의 계층이 깊어질수록 재사용성은 좋으나 이해하기가 어려워지고 유지보수가 힘들어 진다. 반대로 계층이 얕을수록 추상성이 부족하나 이해성과 수정성이 좋아진다. 따라서 시스템의 유지보수성을 위해서는 깊은 상속구조를 분리하여 얕은 상속구조로 만드는 것이 바람직하다. 본 연구에서 제안한 복잡성 척도는 상속구조의 분리에 있어서 복잡도를 정량적으로 평가할 수 있는 정보를 제공한다.

본 연구에서 제안한 척도는 시스템 설계자와 관리자 모두에게 유용하게 사용될 수 있다: 설계자는 개발 과정에 있어서 소프트웨어의 품질을 검토하고 필요한 개선을 하는데 있어서 필요한 정보를 얻을 수 있으며, 관리자는 프로젝트를 평가하고 제어하는데 도움을 얻을 수 있다. 예를 들어 상속을 많이 하여 재사용성을 높일 것인가 아니면 상속계층을 낮게 하여 복잡성을 줄이고 유지보수를 쉽도록 할 것인가에 대한 손익(trade-offs)을 판단하는 등의 정보를 얻을 수 있다.

앞으로의 연구로는 클래스 상속구조를 어떻게 해야 객체지향 시스템의 개발과 유지보수를 쉽게 할 수 있는가하는 근본적 문제와 어느 수준의 상속계층이 최적인가 하는 문제를 해결하는 것이다.

참고문헌

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

[2] N. Balasubramanian, "Object-oriented Metrics," *0-8186-7638-8/96/, IEEE,* pp.30-34, 1996

[3] V. Basili, L. Briand, and W. Melo, "A Validation of Object-Oriented Design Metrics as Quality Indicator," *Technical Report, Univ. of Maryland, Dept. of Computer Science*, College Park, MD, 20742 USA, pp. 1-24, April 1995

[4] 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

[5] L. Briand, J. Wust, J. Daly, and V. Porter, "Exploring the relationships between design measures and software quality on object-oriented systems," *The Journal of Systems and Software*, Vol.51, pp. 245-273, 2000

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

[7] 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

[8] R. Harrison, S. Counsell, and Nithi, "An Evaluation of the MOOD Set of Object-Oriented Software Metrics," *IEEE Transactions on Software Engineering*, Vol.24, No.6, pp. 491-496, 1998

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

[10] B. Kitchenham, S. Pfleeger, and N. Fenton, "Towards a Framework for Software Measurement Validation," *IEEE Transactions on Software Engineering,* Vol.21, No.12, 1995

[11] A. Lake and C. Cook, "Use of Factor Analysis to develop OOP software complexity metrics," *Proc. Workshop on Software Metrics*, Silver Falls, Oregon, 1994

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

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

[14] J. Sherif and P. Sanderson, "Metrics for object-oriented software projects," *The Journal of Systems and Software*, Vol.44, pp. 147-154 , 1998

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

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

[17] C. Wang, T. Shih, and W. Pai, "An Automatic Approach to Object-Oriented Software Testing and Metrics for C++ Inheritance Hierarchies," *International Conference on Information, Communication and Signal Processing, ICICS'97*, Singapore, 9-12 Sep., 1997, pp. 934-938

[18] E. Weyuker, "Evaluating software complexity measures," *IEEE Transactions on Software Engineering*, Vol.14, pp.1357-1365, 1988

저 자 소 개

정 홍(Hong Chung)

1972년 : 한양대학교 원자력공학과(공학사)
1976년 : 고려대학교 경영대학원 생산관리 (경영학석사)
1996년 : 대구카톨릭대학교 전산통계학과 (이학석사)
1999년 : 대구카톨릭대학교 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

김 태 식(Taesik Kim)

1984년 : 계명대학교 컴퓨터공학과
1987년 : Minnesota State University, Moorhead (석사)
1992년 : North Dakota State University (박사)
1992년～현재 : 계명대학교 공학부 컴퓨터공학전공 조교수

관심분야 : Chaos, Neural Networks
E-mail : tkim@kmu.ac.kr