객체 지향 시스템에서의 클래스 응집도와 결합도 메트릭

(Cohesion and Coupling Metric for Classes in Object-Oriented System)

이 종 석 † 우 치 수 ††
(Jong-Seok Lee) (Chi-Su Wu)

요 약 소프트웨어 메트릭스는 개발 과정을 평가하고, 소프트웨어 개발 노력을 측정하며 소프트웨어의 질을 효과적으로 제어할 수 있도록 한다. 더욱이 현재와 같이 재사용성이 강조되고 있는 상황에서는 재사용성을 평가하는데 중요한 역할을 하는 응집도와 결합도에 대한 연구가 반드시 필요하다고 할 수 있다. 캡슐화, 상속, 다형성과 같은 개념을 이용하는 객체 지향 방법론은 기존의 절차적 방법론과는 다른 메트릭스를 요구하는데, 이에 대한 연구가 현재 활발히 진행되고 있다. 본 논문에서는 객체 지향 시스템의 응집도와 결합도를 측정하는 메트릭스를 제안하고, 이를 Weyuker와 Briand의 복잡도 성질을 이용하여 평가하였다. 그리고 C++로 작성된 소프트웨어에 실제 적용하여 응집도와 결합도를 추출하였다.

Abstract Software metrics evaluate the development process, measure the software development effort, and control the software quality effectively. Moreover in a current status to emphasize reusability, it is necessary to study of cohesion and coupling that plays an important role in evaluating reusability. Object oriented methodology to use the concept like encapsulation, inheritance, and polymorphism demands metrics that are different from existing procedural methodology, so a study for object oriented metrics is in progress at the present time. In this paper, we propose cohesion and coupling metrics for object oriented program, evaluate the proposed metrics by using the complexity properties proposed by Weyuker and Briand, and extract cohesion and coupling from C++ code.

1. 서 론

소프트웨어 공학은 질 좋은 소프트웨어 제품을 만드는 방법에 관한 연구를 포함하고 있는데[1], 소프트웨어의 질을 정확히 평가하기 위해서는 그 소프트웨어를 잘 이해하여야 한다. 그런데 DeMarco가 "측정할 수 없는 것은 이해할 수 없다"고 이야기한 것처럼 이해하기 위해서는 반드시 측정할 수 있어야 한다. 여기에 소프트웨어 메트릭스의 필요성이 있다.

· 이 논문은 1997년 한국학술진흥재단의 공모과제 연구비에 의하여 연구되었음
† 종신회원 : 우석대학교 컴퓨터공학과 교수
jong1007@core.woosuk.ac.kr
†† 종신회원 : 서울대학교 전산과학과 교수
wuchisu@selab.snu.ac.kr
논문접수 : 1999년 8월 9일
심사완료 : 2000년 4월 3일

소프트웨어 메트릭스는 개발 과정을 평가하고, 소프트웨어 개발 노력을 측정하며 소프트웨어의 질을 효과적으로 제어할 수 있도록 한다. 그리고 숨겨진 문제를 빨리 찾을 수 있도록 하고, 재사용의 이점을 정량화시킬 수 있도록 한다[2]. 또한 개발자를 위해 자신의 작업의 질을 검사하여 발전시킬 수 있도록 하고, 관리자를 위해 프로젝트 과정을 평가, monitor, control할 수 있도록 한다[3].

기존의 소프트웨어 메트릭스에 관한 연구는 주로 절차적 소프트웨어에 관한 것이었다. 그러나 오늘날에는 절차적 방법론보다 훨씬 빠른 개발과 높은 품질의 소프트웨어를 얻을 수 있는[4] 객체 지향 방법론이 널리 쓰이고 있다. 캡슐화(encapsulatiom), 상속(inheritance), 다형성(polymorphism)과 같은 개념을 이용한 객체 지향 방법론은 기존의 절차적 방법론과는 다른 메트릭스를 요구하는데, 이에 의해 객체 지향 메트릭스에 관한

연구가 현재 활발히 진행되고 있다.

객체 지향 메트릭스에 관한 연구는 90년대 초반부터 본격화되어 지금도 계속 수행 중에 있으며, Whitty가 2000년이 되면 객체 지향 소프트웨어 메트릭스에 관한 논문이 1000 여개에 다다를 것[5]이라고 할 정도로 그동안 수많은 메트릭스가 제안되었다.

기존의 절차적 방법론에서는 하나의 단위 안에 있는 요소들 간의 관계를 재는 척도인 응집도(cohesion)와, 단위들 간의 의존도를 측정하는 척도인 결합도(coupling)가 신뢰성있고 유지 보수가 쉬운 소프트웨어에 관한 중요한 품질 척도였다. 그러나 객체 지향 방법론에서는 강력한 추상화를 제공하여 빠른 시스템 개발과 이해하기 쉬운 소프트웨어를 만들 수 있도록 하였기 때문에 이들 척도에 대한 중요성이 어느 정도 무시되어 왔다. 하지만 개발자의 무지 또는 미숙함으로 인하여 객체 지향 방법론의 장점들을 충분히 살리지 못할 수도 있기 때문에, 단지 객체 지향 방법론만을 도입한다고 해서 좋은 품질의 소프트웨어를 개발할 수 있는 것은 아니다. 그러므로 소프트웨어가 객체 지향 개념에 따라 잘 구성되었는지를 정량화하기 위한 척도들이 점점 더 요구되고 있는 실정이다.

본 논문에서는 이를 해결하기 위한 척도로 클래스의 응집도와 결합도를 제안한다. 응집도를 계산하기 위해 클래스 내에 있는 인스턴스와 메소드의 사용 관계를 조사하여 메소드 연관도와 클래스 응집도를 제안하였고, 결합도를 위해 참조 결합도와 참조 클래스를 제안하였다.

본 논문의 구성은 다음과 같다. 2장에서는 기존의 객체 지향 메트릭스의 응집도와 결합도에 관련된 연구에 대해 살펴보고, 3장에서는 새롭게 정의된 응집도에 대해 설명한다. 4장에서는 새롭게 정의된 결합도에 대해 설명하며, 5장에서는 본 논문에서 제안된 척도들을 C++ 언어로 된 소프트웨어에 실제 작용하여 그 결과를 분석하고, 6장에서는 결론과 향후 연구에 대해 말할 것이다.

2. 관련 연구

2.1 Chidamber의 LCOM과 CBO

[Chi94]에서 응집도로 LCOM(Lack of Cohesion in Methods)을 제안하였다.

LCOM은 모든 메소드간의 조합 쌍 중에서 같은 메소드에 의해 공유되지 않는 인스턴스의 수와 공유되는 인스턴스의 수의 차로써 정의하였다. 그런데 그림 1과 같은 전혀 다른 응집도를 가지는 클래스들이 같은 LCOM 값인 8을 가지는 경우가 생길 수 있다[11]. 또한 LCOM 값이 0일 때 매우 높은 응집도라고 가정되지만, 실제로는 그렇지 않은 경우가 많다. 그리고 클래스에 있는 모든 메소드가 전혀 인스턴스를 사용하지 않는 경우 객체라고 할 수 낮음에도 불구하고 LCOM 값은 0이 된다.

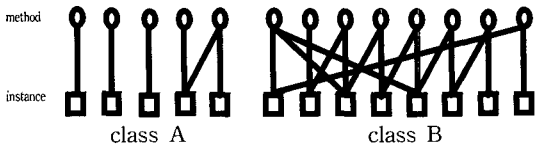

그림 1 LCOM의 예

결합도로는 CBO(Coupling Between Object Classes)를 제안하였는데, CBO는 결합되어 있는 다른 클래스의 수로 정의된다. 여기서 결합되어 있다는 것은 하나의 메소드가 다른 클래스의 메소드나 인스턴스를 사용한다는 것을 의미한다. 이 방법은 하나의 클래스와 관련이 있는 클래스들의 수를 계산함으로써 클래스의 결합도를 간단히 측정할 수 있지만, 한 클래스 내에서 같은 클래스에 있는 멤버들을 여러 번 사용할 때도 한 번 사용한 것과 같은 값을 가진다.

2.2 Wei Li의 응집도와 결합도

[Li93]에서는 응집도를 [Chi94]의 LCOM을 좀더 단순화시켜 지역 메소드의 분리 집합의 개수로 정의하였다. 여기서 분리 집합이란 서로 교집합이 없는 집합을 말하는데, 이 방법은 모든 메소드가 서로 공통된 인스턴스를 하나씩만 가지고 있는 클래스와 모든 메소드들이 아주 밀접하게 연관된 클래스나 모두 다 가장 큰 LCOM 값인 0을 갖는다는 문제점이 있다.

또한 결합도는 세 가지 측면으로 나누어 정의하였다.

결합도의 첫 번째 측면은 상속성을 통한 결합도이다. 객체 지향에서는 재사용성을 위해 상속성을 강조하기 때문에 일반적인 결합도의 성질과는 달리 높을수록 좋다. 하지만 너무 높은 것은 캡슐화와 정보 은닉성을 나쁘게 할 수도 있으므로 적당히 하여야 한다. 이의 측정을 위해 [Chi94]의 DIT와 NOC를 사용하였다.

두 번째 측면은 메시지 패싱을 통한 결합도(MPC : message-passing coupling)이다. 객체 지향 패러다임에서의 통신은 메시지 패싱을 통하여 일어나는데, 이러한 메시지 패싱이 많다는 것은 두 클래스 간의 결합도가 높다는 것을 의미한다. MPC는 CBO를 발전시킨 방법으로 클래스 내에서 정의된 send 문장의 수로 정의하였다. 하지만 한 클래스 내에서 여러 개의 다른 클래스로 메시지를 패싱한 결과와, 한 클래스에게 같은 메시지를 여러 번 패싱한 결과가, 앞의 경우가 결합도가 더 높다는 것을 직관적으로 알 수 있음에도 불구하고 같은

값을 가진다.

세 번째 측면은 추상 데이타 형을 통한 결합도(DAC : data abstraction coupling)이다. 이것은 클래스 내에서 정의된 추상 데이타 형(ADT)의 수로 정의하였는데, 여기에서 ADT의 수가 많다는 것은 다른 클래스의 정의에 연관된 자료 구조가 많다는 것을 의미한다.

2.3 Henderson-Sellers의 응집도와 결합도

[Hen96]에서 LCOM을 발전시켜 "perfect cohesion"인 LCOM*를 다음과 같이 제안하였다.

$$LCOM^* = \frac{(\frac{1}{a}\sum_{j=1}^{a}\mu(A_j)) - m}{1-m}$$

M_i : 메소드의 집합, $1 \le i \le m$

A_j : 인스턴스의 집합, $1 \le j \le a$

$\mu(A_j)$: 각 인스턴스를 사용하는 메소드의 수

그러나 이 방법은 명백히 응집도가 작은 경우에도 더 큰 값을 가질 수 있다. 예를 들어 그림 1에서 class A의 LCOM은 0.95이고, class B의 LCOM은 0.86이다. 이러한 결과가 생기는 이유는 클래스 내에서 분리되어 있는 경우를 고려하지 않았기 때문이다.

결합도로는 [Li93]에서와는 달리 상속에 의한 결합도는 생각하지 않고, DAC, MPC, RFC의 세 가지 측면으로 나누어 제시하였다. [Hen96]에서의 결합도에 대한 정의는 [Li93]에서 정의한 것과 비교하여 상속성을 제외하면 RFC 만이 다르다고 할 수 있다.

DAC는 추상 데이타 형을 통한 결합도로 remote ADT의 수로 결합도를 정의하였는데, 어떤 객체가 다른 객체의 ADT를 여러 번 사용한다고 해도 그 값은 항상 1로서 두 객체 간의 DAC 값은 0이거나 1로 정의하였다.

MPC는 메시지 패싱을 위한 결합도로서, [Li93]에서 정의한 것과 같다.

RFC는 클래스에 있는 모든 메소드의 수에 각 클래스가 호출한 메소드의 수를 합한 것으로 결국 그 클래스에서 사용 가능한 메소드의 수라고 할 수 있다. MPC와의 차이점은 자신의 메소드를 호출할 경우 MPC 값은 증가하지 않지만 RFC는 증가한다는 것이다.

2.4 Lewis의 LC와 CBO

[Lew95]에서는 응집도로 공통으로 사용되는 인스턴스 개수를 고려하여 각 클래스 당 LC(Lack of Cohesion) 값을 다음과 같이 정의하였다.

$$LC = \frac{\sum_{k=1}^{M}\sum_{j=k+1}^{M}|(I_k \cap I_j)|}{1}$$

M : 메소드의 집합

I : 인스턴스 변수의 집합

I_k : 메소드 k가 참조하는 인스턴스 변수의 집합

그런데 이 방법은 가장 응집도가 높은 경우는 0으로 값이 정해져 있지만, 낮은 경우는 메소드의 수에 따라 빠른 속도로 LC의 값이 늘어나므로 메소드의 개수가 다른 경우에는 비교가 거의 불가능하다. 또한 그림 2와 같은 예를 살펴보면 직관적으로 class A의 응집도가 class B의 응집도보다 낮음에도 불구하고, 두 class의 LC 값이 모두 2를 가지는 경우가 생길 수 있다.

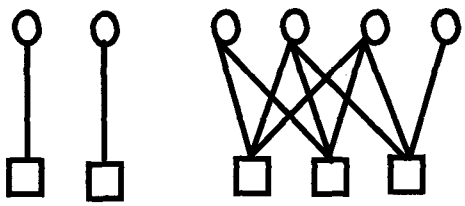

그림 2 LC의 예

결합도로 사용한 CBO는 다음과 같이 정의하였다.

$CBO = EIVR + MS$

$EIVR$: 다른 클래스로부터 *public* 데이타를 액세스하는 인스턴스의 수

MR : 다른 객체에 있는 메소드를 호출하는 메소드의 수

이 척도는 외부 객체에 있는 인스턴스와 메소드의 합으로 결합도를 정의한 것이지만, 같은 인스턴스나 메소드를 여러 번 호출하였을 경우 이를 구별할 수 없다.

2.5 Hitz의 LCOM

[Hitz95]에서는 [Chi94]의 LCOM을 개선하였다. [Chi94]가 제안한 LCOM이 직접 접근하는 인스턴스 변수만을 다루기 때문에 이상 현상이 생긴다고 주장하고, 이를 없애기 위해 메소드가 인스턴스 변수를 사용한다는 의미를 직접 그 인스턴스 변수를 참조하는 것 이외에 간접 참조하는 것도 포함시켰다.

2.6 Biemann의 응집도

[Biem95]에서는 응집도를 측정하는 척도로 TCC (Tight Class Cohesion)와 LCC(Loose Class Cohesion)를 제안하여 직접적으로 연관된 메소드뿐만 아니라 간접적으로 연관된 메소드까지도 고려하였다.

$TCC(C) = NDC(C)/NP(C)$

$LCC(C) = (NDC(C) + NIC(C))/NP(C)$

$NP(C)$: 직접적이거나 간접적으로 연관된 메소드들의 쌍의 수

$NDC(C)$: 직접적으로 연관된 메소드들의 쌍의 수

$NIC(C)$: 간접적으로 연관된 메소드들의 쌍의 수

3. 객체 지향 방법론을 위한 응집도 정의

응집도란 모듈 구성 요소들이 얼마나 연관되었는지를 측정하는 척도이므로 높은 응집도를 가진 모듈이란 하나의 기본적인 기능만을 수행하는 여러 개로 나누어지지 않는 모듈을 의미한다[10]. 그런데 객체 지향 프로그램은 클래스를 단위로 하며, 이 클래스의 구성 요소는 그 클래스 자체에서 정의된 인스턴스 변수와 메소드, 그리고 상속된 요소들이다. 따라서 객체 지향 프로그램의 응집도를 측정한다는 것은 클래스 내에 있는 메소드와 인스턴스 변수들이 얼마나 연관이 있는지 측정하는 것이 될 것이다. 이러한 연관성은 인스턴스 변수들이 메소드에 의해 얼마나 많이 사용되는지에 따라 관련을 맺는다. 즉, 클래스 안의 메소드들이 서로 많은 인스턴스 변수들을 공유한다면 그 클래스는 응집도가 높은 클래스라고 할 수 있을 것이다.

3.1 메소드 연관도

한 클래스의 응집도는 클래스 내의 구성 요소 간 관계를 의미하므로, 잘못 정의된 클래스는 그 응집도가 낮게 측정된다. 그러나 기존의 응집도 척도들은 대부분 두 메소드의 관계들의 합에 의하여 결정하기 때문에 클래스를 전체적인 측면에서 보는 면이 약했다. 그리고 클래스 내에 분리된 부분이 있을 때 응집도로 찾지 못하고 그래프를 이용하여야 찾을 수 있었다. 그러나 큰 시스템에서 각 클래스의 그래프를 그리는 것도 간단한 일이 아닐 뿐 아니라 이를 자동화하는 데에도 문제가 있다.

이를 해결하기 위해 본 논문에서는 먼저 클래스 내의 각각의 메소드가 다른 메소드와 얼마나 연관이 있는지 살펴보는 메트릭스로서 메소드 연관도를 정의한다.

[정의 1] 메소드 연관도(Method Coherency)

$$MC_j = \frac{\sum_{i \in I_j} \mu(i)}{|I_j|}$$

Mi : 메소드의 집합

Ij : 메소드 M_j에 의해 사용되는 인스턴스의 집합

$\mu(i)$: 인스턴스 i를 사용하는 메소드의 수

이와 같이 정의된 척도에 의해 클래스 내에 있는 각 메소드와 다른 메소드들 간의 연관성을 측정할 수 있다. 이 값은 메소드가 다른 메소드와 공유하는 인스턴스 변수가 많으면 그 값이 커지므로, 값이 크면 클수록 그 메소드는 다른 메소드들과 많은 연관성을 가진다고 볼 수 있다.

그리고 어떤 메소드가 다른 메소드와 공유하는 인스턴스가 전혀 없다면 MC 값은 1이 된다. 왜냐하면 MC 값이 1이 되기 위해서는 분자와 분모의 값이 같아야 하는데, 분모는 그 메소드가 사용하는 인스턴스의 수이고, 분자는 그 인스턴스를 사용하는 메소드 수의 합이다. 따라서 그 값이 1이라고 하는 것은 인스턴스를 그 메소드 혼자만이 사용하는 경우이다. 여기에서 MC 값은 항상 1보다 크거나 같다는 것을 알 수 있다.

그림 3에 있는 간단한 예를 통해 각 MC 값을 계산하여 보자.

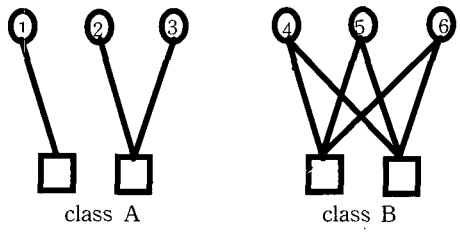

그림 3 MC의 예

$MC_1 = 1$ $MC_2 = 2$ $MC_3 = 2$
$MC_4 = 3$ $MC_5 = 3$ $MC_6 = 3$

그림 3의 클래스에서 볼 수 있듯이, 다른 메소드들과 공유하는 인스턴스가 없이 분리되어 있는 메소드의 MC 값은 1이므로 분리되어 있는 메소드들을 찾을 수 있다. 또한 MC 값이 높은 메소드는 다른 메소드들과 인스턴스를 많이 공유한다는 것을 알 수 있다.

3.2 클래스 응집도

클래스 전체의 응집도를 계산하기 위해서 클래스 응집도를 정의하는데, 이 클래스 응집도는 각 메소드의 MC 값을 합한 값이다. 그러나 단순히 각 메소드의 MC 값을 합칠 경우에는, MC 값이 항상 1 이상이기 때문에 모든 메소드가 독립되어 있는 클래스가 메소드 수에 따라 그 값이 다르게 된다. 따라서 이런 점을 없애기 위해 클래스 응집도를 다음과 같이 정의한다.

[정의 2] 클래스 응집도(Class COherency)

$$CCO = \sum_{j=1}^{m} \tau(j)$$

$$\tau(j) = \begin{cases} 0, & MC_j = 1 \\ MC_j, & otherwise \end{cases}$$

m : 메소드의 수

클래스 응집도의 값은 클수록 그 클래스의 응집도가 좋다는 것을 의미한다. 하지만 일반적으로 복잡도의 값

이 크면 그 클래스가 복잡하다는 것을 나타내기 때문에, 이의 혼동을 피하기 위해 새로운 복잡도 $LCOM^+$를 정의한다. 이 $LCOM^+$는 0과 1 사이의 값을 갖도록 정규화하였다.

[정의 3] $LCOM^+$

$$LCOM^+ = 1 - \frac{CC}{m^2}$$

그림 3의 $LCOM^+$ 값을 계산하면 class A의 값은 0.555이고 class B의 값은 0으로 잘 정의된 클래스의 $LCOM^+$ 값이 작다는 것을 알 수 있다.

3.3 검증

복잡도의 이론적인 검증의 타당성을 증명하기 위해 Weyuker의 복잡도 성질을 많이 이용한다. 물론 Weyuker의 복잡도 성질이 소프트웨어 복잡도에 대한 필요 충분 조건은 아니지만, 현재 이론적 검증을 위해 많이 쓰이고 있으므로, 본 논문에서도 제시한 응집도의 타당성을 Weyuker의 성질을 이용하여 검증한다. 그리고 Weyuker의 성질은 메트릭의 기본 성질을 규명한 것이므로, 이를 보완하기 위해 Briand가 제안한 응집도의 성질을 이용하여 $LCOM^+$를 평가하고자 한다.

3.3.1 Weyuker의 복잡도 성질

응집도를 검증할 때 μ(P)는 클래스 P의 $LCOM^+$라 하고, μ(Q)는 클래스의 Q의 $LCOM^+$라 가정한다. 그리고 μ(P;Q)는 클래스 P와 Q를 합쳤을 때의 $LCOM^+$라 가정한다.

성질 1. $(\exists P)(\exists Q)\ (\mu(P) \neq \mu(Q))$

성질 1은 $LCOM^+$의 값이 다른 클래스가 존재한다는 것을 나타낸다. 이 성질은 다른 $LCOM^+$ 값이 서로 다른 클래스에서 계산된 앞의 예를 통해 쉽게 증명된다.

성질 2. 복잡도가 c인 클래스 또는 프로그램의 수는 유한개이다.

성질 2는 클래스 응집도가 같은 클래스가 유한개임을 나타낸다. 이것은 대부분 적용 범위가 유한 집합이기 때문에 사용되는 클래스의 수가 유한하다는 사실로부터 쉽게 증명된다.

성질 3. $(\exists P)(\exists Q)\ (P \neq Q\ \&\ \mu(P) = \mu(Q))$

성질 3은 응집도는 같지만 다른 기능을 수행하는 클래스가 존재할 수 있다는 의미이다. 이 성질을 증명하기 위해 서로 다른 기능을 수행하는 두 개의 클래스 P, Q가 있다고 가정하자. 클래스 P와 클래스 Q의 내부 요소인 메소드와 인스턴스의 호출 관계가 같다면 각각의 $LCOM^+$의 값은 같다. 따라서 서로 다른 기능을 수행하지만 $LCOM^+$의 값이 같은 클래스가 존재하므로 이 성질을 만족한다.

성질 4. $(\exists P)(\exists Q)\ (P = Q\ \&\ \mu(P) \neq \mu(Q))$

성질 4는 응집도는 다르지만 같은 기능을 수행하는 클래스가 존재할 수 있다는 의미이다. 이 성질을 증명하기 위해 그림 4에 있는 두 개의 클래스를 살펴보자. 두 클래스 stack1과 stack2는 같은 기능을 수행하지만 각각의 $LCOM^+$의 값이 0.111과 0.197이다. 따라서 같은 기능을 수행하지만 구현하는 방법에 따라 클래스의 응집도가 다르기 때문에 이 성질을 만족한다.

```
class stack1 {
        char s[max_len]; int top;
        public:
                void reset() {top=0;}
                void push(char c) {top++; s[top]=c;}
                char pop() {return(s[top--];}
                char top_of() {return(s[top]);}
                boolean empty() {return(top==0);}
                boolean full() {return(top==max_len-1);}
        };
class stack2 {
        char *s; int max_len; int top;
        public:
                stack2(int size) {s=new char[size]; max_len=size;
                top=-1;}
                void reset() {top=-1;}
                void push(char c) {++top; s[top]=c;}
                char pop() {return(s[top--];}
                char top_of() {return(s[top]);}
                boolean empty() {return(top==-1);}
                boolean full() {return(top==max_len-1);}
        };
```

그림 4 성질 4에 관한 예

성질 5. $(\forall P)(\forall Q)\ (\mu(P) \leq \mu(P;Q)\ \&\ \mu(Q) \leq \mu(P;Q))$

성질 5는 임의의 클래스에 새로운 클래스를 삽입하면 $LCOM^+$ 값이 단조 증가한다는 의미이다. 이 성질을 증명하기 위해 그림 5에 있는 클래스들을 생각해 보자. 클래스 1과 클래스 2를 합하여 새로운 클래스 3을 만들었을 때, 클래스 1의 $LCOM^+$ 값은 0.555이고, 클래스 3의 $LCOM^+$ 값은 0.236이다. 즉, 두 클래스를 합쳤을 때의 $LCOM^+$ 값이 더 작아짐을 알 수 있다. 따라서 이 성질은 만족하지 못한다.

성질 6. $(\exists P)(\exists Q)(\exists R)\ (\mu(P) = \mu(Q)\ \&\ \mu(P;R) \neq \mu(Q;R))$

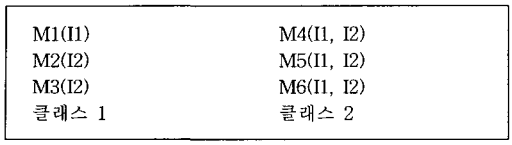

그림 5 성질 5에 관한 예

성질 6은 같은 응집도를 갖는 두 개의 클래스에 새로운 클래스를 앞이나 뒤에 삽입할 경우 응집도가 다르다는 의미이다. 응집도가 같지만 다른 기능을 수행하는 두 개의 클래스 P, Q에 새로운 클래스 R을 각각 뒤에 첨가하였을 경우, 클래스 P와 R 사이에는 같은 메소드가 존재하지만, 클래스 Q와 R 사이에는 같은 메소드가 존재하지 않을 수 있다. 이럴 경우 합해진 클래스들의 $LCOM^+$ 값은 다를 것이므로 성질 6을 만족한다. 클래스 R을 클래스 P와 Q 앞에 삽입하였을 경우도 마찬가지이다.

성질 7. 현재의 클래스 P에서 문장의 순서를 조합하여 새로 만든 프로그램을 Q라 하면, 두 클래스의 응집도는 다르다.

성질 7은 클래스를 구성하는 순서에 따라 응집도가 다르다는 의미이다. 클래스 응집도 $LCOM^+$ 값은 클래스 구성 요소들의 구성 순서나 정의한 순서와 아무런 상관이 없으므로 이 성질은 만족하지 않는다.

성질 8. 클래스 P가 클래스 Q의 변수 이름을 바꾼 클래스라면, 두 클래스 P, Q의 응집도는 같다.

성질 8은 클래스에서 변수 이름을 바꾸어도 클래스 응집도가 변하지 않는다는 의미이다. 클래스를 구성하는 메소드나 인스턴스의 이름을 바꾸어도 클래스 응집도는 변경되지 않기 때문에 성질 8은 만족한다.

성질 9. $(\exists P)(\exists Q)\ (\mu(P) + \mu(Q) < \mu(P;Q))$

성질 9는 두 클래스를 결합하여 하나의 클래스로 만들면 두 클래스 각각의 $LCOM^+$ 값의 합보다 결합한 클래스의 $LCOM^+$ 값이 더 큰 것이 존재한다는 의미이다. 클래스 P, Q를 합하여 새로운 클래스 R을 만들었을 때, 두 개의 클래스 P와 Q가 서로 공통되는 인스턴스와 메소드가 없다면 클래스 R의 $LCOM^+$ 값은 P, Q의 $LCOM^+$ 값을 합한 것보다 더 클 수 있다. 따라서 성질 9를 만족한다.

3.3.2 Briand의 응집도 성질

성질 1. 응집도는 항상 0 이상이고 정규화되어야 한다.

$LCOM^+$는 값이 0에서 1 사이에 있도록 정규화하였으므로 이 성질을 만족한다.

성질 2. 모듈 내에서 관계가 없다면 응집도는 0이다.

클래스에 있는 모든 메소드가 다른 메소드와 공유하는 인스턴스가 전혀 없다면 $LCOM^+$ 값이 1이 되므로 응집도는 0이라고 할 수 있다. 따라서 이 성질을 만족한다.

성질 3. 모듈 내에서 관계를 더하면 응집도가 감소하지 않는다.

메소드와 인스턴스의 수에 변화가 없이 그 관계만 증가한다면 메소드 연관도가 증가하게 되므로 $LCOM^+$ 값이 감소한다. 일반적으로 응집도가 증가한다는 것은 응집도가 좋아진다는 것을 의미하는데, 정의에 의해 $LCOM^+$ 값이 감소하는 것이 좋은 응집도를 나타내므로 이 성질을 만족한다고 할 수 있다.

성질 4. 서로 간에 관계가 없는 두개의 모듈을 합쳤을 경우 응집도는 증가하지 않는다.

서로 연관이 없는 즉, 같은 인스턴스나 메소드를 사용하지 않는 두 개의 모듈을 합치면 클래스 응집도는 두 모듈의 클래스 응집도를 합친 것과 같지만, $LCOM^+$는 이를 메소드 개수의 제곱으로 나누어 계산하기 때문에 각각의 $LCOM^+$ 값보다 증가하게 된다. 따라서 응집도가 감소하므로 이 성질을 만족한다.

본 연구에서 제안한 응집도는 Weyuker의 성질 5와 7만 만족하지 못하고, Briand의 응집도 성질은 모두 만족한다. 따라서 다른 응집도들이 만족하는 성질들을 최소한 만족하므로 응집도로서 적합하다고 볼 수 있다.

4. 객체 지향 방법론을 위한 결합도 정의

결합도란 모듈이 다른 모듈과 얼마나 밀접하게 연관되어 있는지를 측정하는 척도이므로, 낮은 결합도를 가진 모듈이 다른 모듈에 영향을 받지 않고 독립적인 기능을 수행할 수 있다. 그러나 객체 지향 프로그램은 재사용성을 위해 상속성을 강조하기 때문에 상속성을 통한 결합도는 높을수록 좋다고 할 수 있다. 하지만 상속성을 통한 결합도가 너무 높다면 일반적인 의미의 결합도에서와 마찬가지로 캡슐화와 정보 은닉성을 나쁘게 한다. 더구나 상속을 받는다는 것은 부모 클래스의 성질을 이해해야 하기 때문에 좋은 의미의 결합도라고 하더라도 클래스를 이해하는데 어려움을 준다. 따라서 상속성을 통한 결합도도 다른 의미의 결합도와 마찬가지로 클래스의 복잡도를 높인다고 할 수 있다.

그러므로 본 논문에서는 상속성을 통한 결합도도 다

른 결합도와 같이 취급하여 클래스의 결합도를 측정하고자 한다.

4.1 참조 결합도

한 클래스의 결합도를 알기 위해서는 그 클래스가 다른 클래스들과 어떤 연관을 갖고 있는지 측정해야 한다. 그런데 결합도란 본래 두 모듈간의 관계를 의미하므로, 클래스의 결합도를 측정하기 위해서는 먼저 각 클래스간의 결합도를 알아야 한다. 이를 위해 외부 참조 요소를 정의하는데, 이것은 각각의 메소드가 참조하는 외부 클래스에 있는 요소들의 수를 나타낸다.

[정의 4] 외부 참조 요소(External Reference Elements)

$$ERE(m_i, O) = \sum \eta(m_i, O)$$

$$\eta(m_i, O) = \begin{cases} 1, & \psi(m_i) \in O \\ 0, & otherwise \end{cases}$$

$\psi(m_i)$: 메소드 m_i에 의해 사용되는 인스턴스와 메소드

즉, 클래스에서 정의된 메소드 m_i에 의해 참조되는 요소 중 외부 클래스 O에 있는 인스턴스와 메소드의 수를 외부 참조 요소 $ERE(m_i, O)$라고 정의한다. 이 외부 참조 요소에 의해 한 클래스 내의 메소드가 다른 클래스에 있는 요소들을 얼마나 많이 사용하는지 알 수 있다.

잘 정의된 클래스는 결합도가 낮은데, 이는 외부 클래스의 요소들을 참조하는 율이 낮다는 것을 의미한다. 따라서 클래스의 결합도를 측정하기 위해서는 그 클래스가 클래스 내부의 요소와 외부의 요소들을 얼마나 참조하는지 그 정도를 알아야 한다. 이를 위해 먼저 클래스쌍 의존도를 정의한다. 클래스쌍 의존도는 외부 클래스에 연관을 갖는 모든 메소드에 대한 외부 참조 요소의 값을 합한 것을 그 클래스가 참조하는 모든 요소들의 수로 나눈 것으로, 클래스쌍 의존도가 높다는 것은 상대 클래스에 있는 요소들을 많이 사용한다는 의미이다. 클래스쌍 의존도는 한 클래스와 결합 관계가 높은 클래스를 찾는 데 이용할 수 있으므로, 이를 통해 클래스를 재구성할 수 있다.

[정의 5] 클래스쌍 의존도(Object-Pair Dependency)

클래스 O_1에서 O_2로의 클래스쌍 의존도를 다음과 같이 정의한다.

$$OPD(O_1, O_2) = \frac{\sum_{m_i \in O_2} ERE(m_i, O_2)}{\sum_{m_i \in O_2} \eta(m_i, O_1) + \sum_j \sum_{m_i \in O_j} ERE(m_i, O_j)}$$

O_j : O_1이 참조하는 모든 외부 클래스

한 클래스와 연관이 있는 모든 클래스와의 클래스쌍 의존도를 합한 결과를 그 클래스의 참조 결합도라고 정의한다. 이 참조 결합도는 외부 클래스의 요소를 참조하는 정도를 나타내는 척도로 외부 클래스의 요소를 참조하는 율이 크면 클수록 1에 가까워진다.

[정의 6] 참조 결합도(Reference Coupling)

클래스 C의 참조 결합도를 다음과 같이 정의한다.

$$RC(C) = \sum OPD(C, O_i)$$

O_i : 클래스 C에서 참조하는 모든 외부 클래스

4.2 클래스 결합도

많은 수의 외부 클래스를 참조하지만 각 클래스 내의 요소들을 상대적으로 조금만 사용하는 클래스가 있다면 참조 결합도의 값은 작을 것이다. 예를 들어 그림 6에서 클래스 A의 참조 결합도는 0.308인데 반하여 클래스 B의 참조 결합도는 0.333이므로 클래스 A의 결합도가 낮다. 하지만 클래스 A는 참조하는 외부 클래스의 수가 많기 때문에 오히려 더 이해하기 힘들어지기 때문에 결합도도 높다고 할 수 있다.

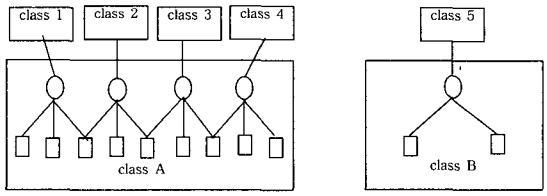

그림 6 참조 결합도의 예

따라서 이러한 점을 보안하기 위해 참조 클래스를 정의한다. 이 참조 클래스에는 [Li93]에서 정의한 결합도에 대한 모든 경우의 참조를 포함한다.

[정의 7] 참조 클래스(Reference Class)

$$RC(A) = |\rho(A)|$$

$\rho(A)$: 클래스 A가 참조하는 외부 클래스

클래스의 결합도는 외부 클래스의 요소를 참조하는 비율과 함께 참조하는 외부 클래스의 수를 모두 고려하여야 한다. 그러므로 참조 결합도와 참조 클래스를 곱한 것을 클래스의 결합도라고 정의한다.

[정의 8] 클래스의 결합도(Coupling of Class)

$$CC(A) = RC(A) * RO(A)$$

4.3 검증

응집도에서와 마찬가지로 Weyuker의 복잡도 성질과 Briand의 결합도 성질을 이용하여 본 논문에서 제안한 CC를 평가하고자 한다.

4.3.1 Weyuker의 복잡도 성질

결합도를 검증할 때 v(P)는 클래스 P의 CC라 하고, v(Q)는 클래스의 Q의 CC라 가정한다. 그리고 v(P;Q)는 클래스 P와 Q를 합쳤을 때의 CC라 가정한다.

성질 1. (∃P)(∃Q) (v(P) ≠ v(Q))

성질 1은 결합도 값이 다른 클래스가 존재한다는 것을 나타낸다. 이 성질은 다른 결합도 값이 서로 다른 클래스에서 계산된 그림 6의 예를 통해 쉽게 증명된다.

성질 2. 복잡도가 c인 클래스 또는 프로그램의 수는 유한 개이다.

성질 2는 클래스 결합도가 같은 클래스가 유한 개임을 나타낸다. 이것은 앞에서와 같이 대부분 적용 범위가 유한 집합이기 때문에 사용되는 참조 요소의 수와 클래스의 수가 유한하다는 사실로부터 쉽게 증명된다.

성질 3. (∃P)(∃Q) (P ≠ Q & v(P) = v(Q))

성질 3은 결합도는 같지만 다른 기능을 수행하는 클래스가 존재할 수 있다는 의미이다. 이 성질은 증명하기 위해 서로 다른 기능을 수행하는 두 개의 클래스 P, Q가 있다고 가정하자. 클래스 P와 클래스 Q가 같은 수의 외부 클래스를 참조하면서 클래스쌍 의존도들도 같다면 각각의 CC의 값은 같다. 따라서 서로 다른 기능을 수행하지만 CC의 값이 같은 클래스가 존재하므로 이 성질을 만족한다.

성질 4. (∃P)(∃Q) (P = Q & v(P) ≠ v(Q))

성질 4는 결합도는 다르지만 같은 기능을 수행하는 클래스가 존재할 수 있다는 의미이다. 이 성질을 증명하기 위해 같은 기능을 수행하는 두 개의 클래스 P, Q가 있다고 가정하자. 클래스 P는 내부에 있는 요소만을 참조하고, 클래스 Q는 외부 클래스의 요소를 참조한다고 한다면 두 클래스의 결합도가 달라지게 되므로 이 성질을 만족한다.

성질 5. (∀P)(∀Q) (v(P) ≤ v(P;Q) & v(Q) ≤ v(P;Q))

성질 5는 임의의 클래스에 새로운 클래스를 삽입하면 CC 값이 단조 증가한다는 의미이다. 이 성질을 증명하기 위해 그림 6의 클래스 A와 같은 클래스 P와, 외부 참조 요소가 없이 단지 내부 요소 3개만을 참조하는 메소드 하나를 가진 클래스 Q를 가정하면, 클래스 P의 CC 값은 1.231 이고, 클래스 Q의 CC 값은 0이다. 그런데 클래스 P와 Q를 합하여 새로운 클래스 R을 만들었을 때, 클래스 R의 CC 값은 1이므로 클래스 P의 값보다 더 작아짐을 알 수 있다. 따라서 이 성질은 만족하지 못한다.

성질 6. (∃P)(∃Q)(∃R) (v(P) = v(Q) & v(P;R) ≠ v(Q;R))

성질 6은 같은 결합도를 갖는 두 개의 클래스에 새로운 클래스를 앞이나 뒤에 삽입할 경우 결합도가 다르다는 의미이다. 결합도가 같지만 다른 기능을 수행하는 두 개의 클래스 P, Q에 새로운 클래스 R을 각각 뒤에 첨가하였을 경우, 클래스 P와 R은 같은 클래스를 참조하지만, 클래스 Q와 R은 같은 클래스를 참조하지 않을 수 있다. 이럴 경우 합해진 클래스들의 CC 값은 다를 것이므로 성질 6을 만족한다. 클래스 R을 클래스 P와 Q 앞에 삽입하였을 경우도 마찬가지이다.

성질 7. 현재의 클래스 P에서 문장의 순서를 조합하여 새로 만든 프로그램을 Q라 하면, 두 클래스의 결합도는 다르다.

성질 7은 클래스를 구성하는 순서에 따라 결합도가 다르다는 의미이다. 클래스 결합도 CC 값은 클래스 구성 요소들의 구성 순서나 정의한 순서와 아무런 상관이 없으므로 이 성질은 만족하지 않는다.

성질 8. 클래스 P가 클래스 Q의 변수 이름을 바꾼 클래스라면, 두 클래스 P, Q의 결합도는 같다.

성질 8은 클래스에서 변수 이름을 바꾸어도 클래스 결합도가 변하지 않는다는 의미이다. 클래스를 구성하는 메소드나 인스턴스의 이름을 바꾸어도 클래스 결합도는 변경되지 않기 때문에 성질 8은 만족한다.

성질 9. (∃P)(∃Q) (v(P) + v(Q) < v(P;Q))

성질 9는 두 클래스를 결합하여 하나의 클래스로 만들면 두 클래스 각각의 CC 값의 합보다 결합한 클래스의 CC 값이 더 큰 것이 존재한다는 의미이다. 클래스 P, Q를 합하여 새로운 클래스 R을 만들었을 때, 두 개의 클래스 P와 Q에 있는 메소드가 같은 내부 요소를 참조하지만 다른 외부 요소를 참조하면 클래스 R의 CC 값이 두 개의 클래스의 CC 값을 합한 것보다 항상 작거나 같다. 따라서 성질 9를 만족하지 못한다.

4.3.2 Briand의 결합도 성질

성질 1. 결합도는 0 이상이다.

클래스 결합도 CC 값은 정의에 의하여 음수 값이 나올 수 없으므로 이 성질을 만족한다.

성질 2. 모듈 간에 관계가 없다면 결합도는 0이다.
참조하는 클래스가 아무 것도 없다면 CC 값이 0이 되므로 이 성질을 만족한다.

성질 3. 모듈 간에 관계를 더하면 결합도는 감소하지 않는다.
두 클래스간에 새로운 관계가 더해지면 관계가 더해진 메소드의 외부 참조 요소가 증가하고, 이에 참조 결합도가 증가하게 되므로 CC 값이 증가한다. 따라서 이 성질을 만족한다.

성질 4. 두 개의 모듈을 합쳐서 만든 모듈의 결합도는 두 모듈의 결합도의 합보다 작거나 같다.
두개의 클래스가 합쳐지면 각 메소드의 외부 참조 요소는 줄거나 같게 된다. 그리고 합쳐진 클래스가 참조하는 모든 요소들의 수는 각 클래스가 참조하는 모든 요소들의 수를 합친 것과 같으므로 클래스 쌍 의존도는 작아지고, 그 합인 참조 결합도는 각 클래스의 참조 결합도의 합보다 작다. 또한 참조 클래스의 수도 두 클래스의 참조 클래스의 수를 합친 것보다 작거나 같게 된다. 따라서 합쳐진 클래스의 CC 값은 두 클래스의 CC 값의 합보다 작게 되므로 이 성질을 만족한다.

성질 5. 두 개의 모듈이 서로 관계가 없을 경우 두 모듈을 합쳐서 만든 모듈의 결합도는 두 모듈의 결합도의 합과 같다.
두개의 클래스가 서로 공유하는 클래스가 없을 경우에 참조 결합도는 각 클래스의 참조 결합도의 합보다 작고, 참조 클래스의 수는 각 클래스의 참조 클래스를 합한 것과 같다. 따라서 합쳐진 클래스의 CC 값은 두 클래스의 CC 값의 합보다 작게 되므로 이 성질을 만족하지 못한다.

본 연구에서 제안한 결합도는 Weyuker의 성질 5와 7, 9를 만족하지 못하고, Briand의 결합도 성질 5를 만족하지 못한다. 하지만 다른 결합도들이 만족하는 성질들을 최소한 만족하므로 결합도로서 적합하다고 볼 수 있다.

5. 실험 및 평가

본 연구에서 제안한 응집도와 결합도를 분석하기 위해 C++ 언어로 작성된 프로그램을 이용하여 테스트하였다. 이 프로그램은 객체 지향 설계를 위한 개발 환경 시스템으로, 총 라인 수는 24125, 클래스의 수는 122개이며, 총 인스턴스와 메소드의 수는 각각 760, 1347개이다.

그림 7과 그림 8은 각각 LCOM과 LCOM$^+$에 대한 분포도이다. 두 개의 도표 모두 이 시스템의 응집도가 좋지 않음을 나타내고 있다.

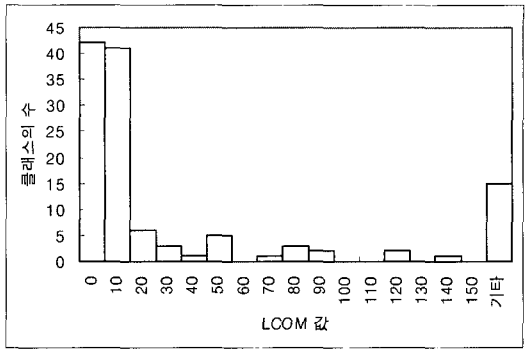

그림 7 LCOM의 분포도

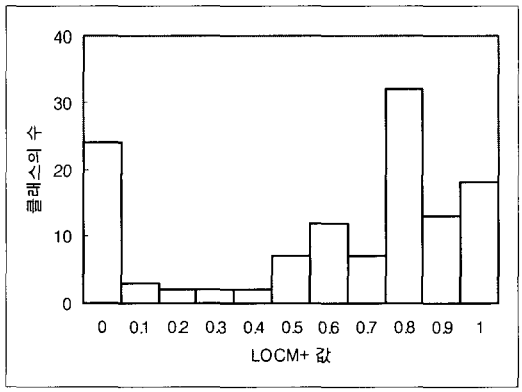

그림 8 LCOM+의 분포도

표 1은 LCOM과 LCOM$^+$를 비교한 결과이다.

표 1 LCOM과 LCOM+ 비교

	Min	Max	Median
LCOM	0	2010	3
LCOM$^+$	0	0.983	0.75

표 1에서 LCOM의 최대값은 2010 임을 볼 수 있는데, 이는 메소드의 개수가 150 개인 굉장히 큰 클래스로, 각 메소드가 공통되는 인스턴스를 거의 사용하지 않고 있음을 알 수 있다. 이 클래스의 LCOM$^+$ 값은 0.983으로 역시 큰 값을 나타내지만, 그 편차가 LCOM 만큼 크지는 않다. 또한 클래스 내에 있는 메소드의 수가 늘

어남에 따라 LCOM 값은 큰 폭으로 늘어나는 경향이 있는 반면, $LCOM^+$ 값은 정규화를 시켰기 때문에 0과 1 사이에 존재한다.

그리고 LCOM 값이 0인 클래스들의 $LCOM^+$ 값은 0에서 0.56까지 분포되어 있는데, 이는 대부분의 메소드가 밀접하게 연관되어 있다고 하더라도 다른 메소드와 공통되는 인스턴스를 갖지 않는 메소드가 존재할 경우 이 클래스의 LCOM 값은 0이 되는 반면, $LOCM^+$ 값은 0이 되지 않고 0.5에 근접한 값을 가지게 되기 때문이다.

또한 두 클래스의 LCOM과 $LCOM^+$ 값의 크기 순서가 서로 바뀌는 경우도 볼 수 있다. 그림 9에서와 같이 직관적으로 클래스 A의 응집도가 작음에도 불구하고, 클래스 A와 B의 LCOM 값은 각각 4와 7로 클래스 A의 응집도가 크게 나타난다. 그러나 $LCOM^+$ 값은 각각 0.781과 0.701로 클래스 A의 응집도 값이 작다. 이러한 현상이 나타나는 이유는 $LCOM^+$는 LCOM에서와 같이 클래스 내의 응집도를 단순히 메소드의 쌍으로 보는 것이 아니라, 전체 클래스의 관점에서 보았기 때문이다. 따라서 $LOCM^+$ 값이 LCOM 값보다 더 정확한 응집도 값을 나타낸다고 할 수 있다.

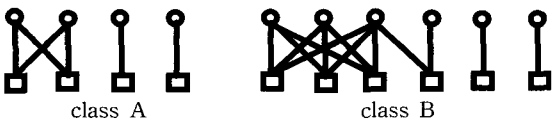

그림 9 LCOM과 LCOM+의 비교

이외에도 MC 값이 1인 메소드는 클래스 내에서 공통적으로 참조하는 인스턴스가 없이 독립적으로 존재하는 것임을 나타내는 것이므로, 이를 통해 클래스를 재구성하는 데에도 도움을 줄 수 있다.

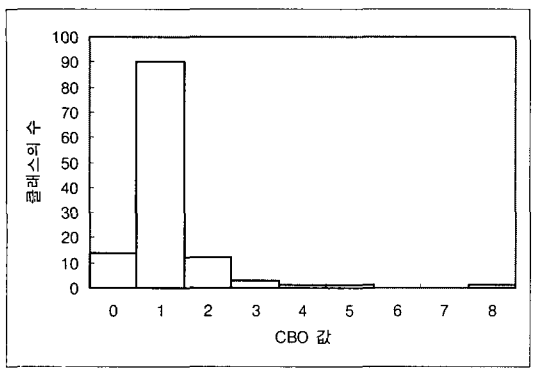

그림 10 CBO의 분포도

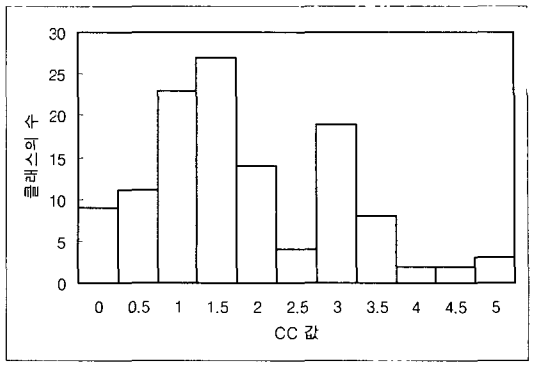

그림 11 CC의 분포도

표 2 CBO 와 CC 비교

	Min	Max	Median
CBO	0	8	1
CC	0	5	1.5

그림 10과 그림 11은 각각 CBO와 CC에 대한 분포도이고, 표 2는 CBO와 CC를 비교한 결과이다.

이 시스템은 CBO와 CC의 median 값이 각각 1과 1.5이므로 결합도가 좋다고 할 수 있다. 그러나 CBO는 2보다 큰 값은 거의 없지만, CC는 0에서 3까지 거의 비슷하게 분포되어 있음을 알 수 있다. 이렇게 두 값이 분포에 있어서 차이가 나는 이유는 CBO가 외부 클래스의 인스턴스와 메소드를 사용하는 경우만 포함시킨 반면, CC는 추상 데이타를 통하여 결합된 것도 포함하는 참조 클래스를 같이 사용하기 때문이다. 또한 이 시스템에서 CBO와 CC의 상관 관계는 응집도에서와는 달리 굉장히 높지만, 몇 개의 클래스에서 다른 결과를 나타내는 경우가 있는데, 그러한 경우는 CBO의 값은 작지만 CBO 이외의 참조하는 외부 클래스가 많은 경우이다. 따라서 CC의 값이 CBO의 값보다 좀더 정확한 결합도를 나타낸다고 할 수 있다.

그리고 CC는 참조 클래스 외에 메소드가 참조하는 전체 요소에서 외부 클래스의 요소를 참조하는 정도를 나타내는 참조 결합도를 사용하기 때문에, 이를 이용하면 외부에 있는 클래스와 결합 관계가 높은 클래스를 찾아 좀 더 낮은 결합도를 가진 클래스로 재구성 할 수 있다.

이 시스템의 RC에 대한 분포도는 그림 12와 같다. 이 분포도를 CBO와 관련해서 보면 각 클래스들이 외부 클

래스의 메소드나 상속을 이용하는 것보다는 추상 데이타 형태로 많이 이용한다는 것을 알 수 있다.

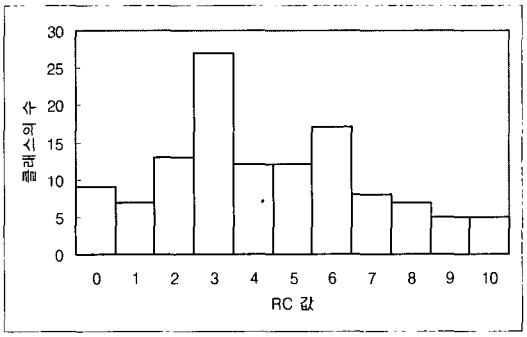

그림 12 RC의 분포도

7. 결 론

본 논문에서는 객체 지향 시스템의 성능 측정을 위한 응집도와 결합도 메트릭스를 제안하였다. 클래스 내의 각각의 메소드가 다른 메소드와 얼마나 연관되어 있는지 측정하는 메소드 연관도를 이용해 각 메소드와 다른 메소드와의 연관성을 살피고, 이를 정규화함으로써 각 클래스의 응집도를 측정하는 척도로 이용하였다. 또한 각 클래스가 참조하는 외부 클래스에 있는 요소들의 수를 합한 값을 그 클래스가 참조하는 모든 요소의 수로 나눈 참조 결합도와, 참조하는 외부 클래스의 수인 참조 클래스를 곱한 값을 각 클래스의 결합도를 측정하는 척도로 이용하였다.

본 연구에서 제안한 응집도와 결합도 메트릭은 Weyuker가 제안한 복잡도 척도의 성질을 최소한 만족하고, Briand가 제안한 응집도 성질은 모두 만족한다. 그리고 결합도에서는 성질 5만 제외하고는 모두 만족한다. 또한 이 메트릭들을 실제 코드에 적용하여 [Chi94]의 LCOM, CBO와 비교한 결과, 그 값들보다 정확하게 응집도와 결합도를 표현하였다. 이외에도 메소드 연관도를 이용하면 분리되어 있는 메소드를 찾을 수 있고, 클래스쌍 의존도를 이용하면 한 클래스와 결합 관계가 높은 클래스를 찾을 수 있어 클래스를 재구성하는 데에도 도움을 받을 수 있다.

앞으로의 연구 과제는 본 연구에서 제안한 응집도와 결합도를 더 많은 실무 데이타를 이용한 실험을 통하여 그 타당성을 입증하고, 각 클래스의 응집도와 결합도가 아닌, 컴퍼넌트의 응집도와 결합도를 측정할 수 있는 메트릭을 제안하는 것이다.

참 고 문 헌

[1] W. Li, S. Henry, "Object-Oriented Metrics That Predict Maintainability," J. System SW, Vol 23, pp.111-122, 1993.

[2] F. B. e Abreu, R. Carapuca, "Candidate Metrics for Object-Oriented Software within a Taxonomy Framework," J. System SW, Vol 26, pp.87-96, 1994.

[3] V. B. Misic, S. Moser, "Measuring Class Coupling and Cohesion : A Formal Metamodel Approach," APSEC'97, pp.31-40, Dec., 1997.

[4] J. A. Lewis, "Quantified Object-Oriented Development: Conflict and Resolution," Proc. 4th SW Quality Conf. Vol 1, pp.220-229, 1995.

[5] R. Whitty, "Object-Oriented Metrics : A Status Repor,t, Object Expert, pp.35-40, Jan-Feb, 1996.

[6] S. R. Chidamber, C. F. Kemerer, "A Metrics Suite for Object Oriented Design," IEEE Tr. on SE, Vol. 20, No. 6, pp.476-493, 1994.

[7] B. Henderson-Sellers, "Identifying Internal and External Characteristics of Classes Likely to Be Useful As Structural Complexity Metrics," Proc. 1994, Intl. Conf. OO Information Systems, OOIS'94, pp.227-230, Dec., 1994.

[8] M. Hitz, B. Montazeri, "Measuring Product Attributes of Object-Oriented Systems," Proc. ESEC'95, pp.1-13, 1995.

[9] M. Hitz, B. Montazeri, "Measuring Coupling and Cohesion in Object-Oriented Systems," Proc. Intl. Symp. Applied Corporating, pp.75-84, Oct., 1995.

[10] B. K. Kang, J. M. Bieman, "Cohesion and Reuse in an Object-Oriented Systems," Proc. ACM SW Reusability Symp. SE Notes, pp.1-4, Aug., 1995

[11] B. Henderson-Sellers, Object-Oriented Metrics Measures of Complexity, Pretice Hall, 1996.

[12] E. J. Weyuker, "Evaluation Software Complexity Measures," IEEE Transactions on SE, Vol.14, NO.9, pp.1357-1365, 1988.

[13] R. C. Sharble, S. S. Cohen, "The Object-Oriented Brewery: A Comparison of Two Object-Oriented Development Methods," ACM Sigsoft SE Notes, Vol 18, No 2, pp.60-73, Apr., 1993.

[14] L. Briand, S. Morasca, V. R. Basili, "Property-Based Software Engineering Measurement," IEEE Transactions on SE, Vol.22, NO.1, pp.68-85, Jan. 1996.

[15] Sunghee Park, Pyeongsoo Mah, Gyusang Shin, "Metrics Measuring Cohesion and Coupling to Compare Object Models," TOOLS Pacific'97, pp.317-323, 1997.

[16] 박성희, 홍의석, 이종석, 우치수, "객체 지향 개념에서의 응집도 측정을 위한 메트릭스," 한국정보과학회 학술발표논문집(B), 제23권, 2호, pp.1503-1506, Oct., 1996.

[17] 김갑수, 신영길, 우치수, "엔트로피를 이용한 객체 지향 프

로그램의 복잡도 척도", 정보과학회논문지(B), 제22권, 제12호, pp1656-1666, 1995.
[18] 김철진, 조은숙, 김수동, "효율적인 객체지향 설계 및 성능 측정을 위한 정적/동적 메트릭", 정보과학회논문지(B), 제25권, 제11호, pp.1657-1666, 1998.

이 종 석

1988년 서울대학교 계산통계학과 학사. 1990년 서울대학교 계산통계학과 석사. 1991 ~ 현재 서울대학교 전산과학과 박사 과정. 1993년 ~ 현재 우석대학교 컴퓨터공학과 조교수. 관심분야는 소프트웨어 공학, 소프트웨어 복잡도, 컴퍼넌트 소프트웨어

우 치 수

정보과학회논문지 : 소프트웨어 및 응용
제 27 권 제 2 호 참조