의존 인스턴스 변수를 고려한 클래스 응집도 척도의 개선

(Improving Cohesion Metrics for Classes By Considering Dependent Instance Variables)

채 흥 석 † 권 용 래 †† 배 두 환 ††
(Heung Seok Chae) (Yong Rae Kwon) (Doo Hwan Bae)

요 약 응집도는 모듈의 구성 요소들 사이의 관련성 정도를 나타내는 척도로서, 응집도가 높을수록 소프트웨어에 대한 이해 및 유지보수가 용이하다고 알려져 있다. 최근에 응집도의 개념을 객체지향 시스템의 클래스에 적용하기 위하여 많은 응집도 척도들이 제안되고 있다.

그러나, 기존의 응집도 척도들은 다른 인스턴스 변수의 값에 의존하는 의존 인스턴스 변수의 특성을 고려하지 않았기 때문에 클래스의 응집도를 정확하게 측정하지 못하였다. 본 논문에서는 응집도 평가 시 의존 인스턴스 변수에 대한 고려를 통하여 기존의 응집도 척도를 개선시킬 수 있음을 소개한다. 그리고, 의존 인스턴스의 특성을 기존의 응집도 척도에 반영한 실험을 통하여 의존 인스턴스 변수에 대한 고려의 중요성을 소개한다.

키워드 : 객체지향 시스템, 클래스 응집도

Abstract Cohesion refers to the degree of the relatedness of the elements in a module, and it is widely accepted that the module of higher cohesion is easier to understand, maintain, and reuse. Recently, several cohesion metrics have been proposed to measure the cohesiveness of classes in an object-oriented program. However, the existing cohesion metrics do not consider the characteristics of dependent instance variables that are commonly used in a class and, thus, do not properly reflect the cohesiveness of the class. This paper presents an approach for improving the cohesion metrics by considering the characteristics of the dependent instance variables in an object-oriented program. To demonstrate the importance of the dependent instance variables, a case study has been conducted on a class library.

Key words : object-oriented systems, class cohesion

1. 서 론

구조적 설계 기법에서 도입되었던 응집도 개념은 모듈의 구성 요소들 사이의 관련성 정도를 나타내는 척도이다[1]. 높은 응집도의 모듈은 오직 하나의 기능 만을 제공하며 모든 구성 요소들이 이 기능을 구현하기 위하여 서로 밀접하게 상호작용을 한다. 반면에, 낮은 응집도의 모듈은 서로 관련성이 적은 구성 요소들로 구성이 되고, 여러 기능을 제공한다. 일반적으로 응집도가 높은 모듈은 개발, 유지보수, 재사용이 용이하다고 알려져 있다[2,3]. 따라서, 시스템을 설계할 때 가능한 한 높은 응집도의 모듈로서 설계가 될 수 있도록 하며, 유지 보수 단계에서도 높은 수준의 응집도를 유지할 수 있도록 노력을 하는 것이 바람직하다. 이제까지 모듈의 응집도를 측정하기 위한 많은 척도들이 제안되었으며 이들 척도들은 개발자들이 더 높은 응집도의 모듈을 개발할 수 있도록 사용되고 있다.

객체지향 시스템은 실 세계에 존재하는 물리적인 개체 및 추상적인 개념을 객체/클래스로서 모델링 함으로써 기존의 개발 방법에 비하여 보다 높은 생산성과 유지보수성을 제공하는 것으로 인정되고 있다. 클래스는 실 세계의 중요한 대상을 추상화 시킨 모델로서 객체지향 시스템의 개발에서 가장 중요한 역할을 한다. 즉, 실

· 본 연구는 대학 IT 연구 센터 육성 지원 사업의 연구 결과로 수행되었음
† 정 회 원 : 부산대학교 컴퓨터 공학과 교수
hschae@pusan.ac.kr
†† 종신회원 : 한국과학기술원 전자전산학과 교수
kwon@selab.kaist.ac.kr
bae@selab.kaist.ac.kr
논문접수 : 2004년 1월 30일
심사완료 : 2004년 6월 25일

세계에 존재하는 중요한 대상은 클래스로서 모델링 되고, 시스템은 이들 클래스로부터 생성된 객체들에 의해서 구축된다. 클래스는 인스턴스 변수와 메소드로 구성된다. 인스턴스 변수는 객체의 상태에 대한 정보를 나타내며 메소드는 객체가 제공하는 기능을 정의한다. 객체지향 시스템의 클래스는 정보은닉의 단위가 된다. 즉, 클래스의 인스턴스(객체)는 반드시 클래스에 정의된 인터페이스를 통해서만 조작될 수 있다. 따라서, 기존의 인터페이스를 만족시키는 범위 내에서 클래스의 내부(인스턴스 변수, 메소드의 구현)의 변경은 시스템의 다른 부분에 전혀 영향을 미치지 않으므로 시스템의 유지보수성 및 확장성에 큰 기여한다[4].

객체지향 시스템이 제공하는 이점의 성취 여부는 클래스의 품질에 의존한다. 따라서, 객체지향 방법론들은 고품질의 클래스를 정의하기 위한 다양한 가이드라인을 제공하고 있다. 그러나, 분석 및 설계 단계에서 클래스의 개념을 부적절하게 적용한다든가 또는 유지보수 단계에서 무분별하게 프로그램을 수정하는 경우에 낮은 품질의 클래스들이 발생할 위험이 있다. 따라서, 클래스의 품질에 대한 평가와 함께 낮은 품질의 클래스에 대해서는 재구성 작업이 수반되어야 한다.

구조적 설계 방법에서처럼 응집도는 낮은 품질의 클래스를 파악하는 데 사용될 수가 있다. 즉, 클래스가 실세계의 대상에 대한 적절한 표현이 아니고, 관련성이 적은 멤버들의 단순한 집합이라면 낮은 응집도를 보일 것이다. 반대로, 클래스가 객체의 특성을 잘 기술하고 있다면, 클래스의 멤버들은 높은 관련성을 가질 것이고, 따라서 높은 응집도를 보일 것이다. 실제로 많은 객체지향 방법론들은 클래스의 설계에 대한 품질을 평가하는 방법으로서 응집도를 권장하고 있다.

객체지향 시스템에 대한 관심과 더불어 클래스의 응집도를 측정하기 위한 척도들이 많이 제안되고 있다[5-12]. 초기의 응집도 척도들은 구성 요소 간의 관련성 정도라는 응집도의 기본 개념만을 바탕으로 하였기 때문에 기존의 모듈과는 다른 클래스 고유의 특성을 반영하지 못하는 문제점을 가지고 있음이 지적되었다[5-7]. 예를 들어, Bieman 등[5]은 공용 메소드와 비공용 메소드를 구분하고, 공용 메소드들만의 관련성 정도로서 LCC (Loose Class Cohesion)과 TCC(Tight Class Cohesion) 척도를 제안하였다. 또한, 클래스의 생성자와 소멸자에 의해서 발생할 수 있는 문제점을 해결하기 위하여 생성자와 소멸자를 응집도의 측정 시 제외하였다. Briand 등[6]은 접근 메소드[1]가 클래스의 응집도를 불필요하게 약화시키는 문제점을 지적하고 이들 접근 메소드를 제외하고 응집도를 측정해야 한다고 주장하였다. 또한, CBMC(Cohesion Based on Member Connectivity)[7]에서는 접근 메소드, 생성자, 소멸자와 더불어 위임 메소드도 동일한 문제를 유발할 수 있고, 이를 특별 메소드라는 개념을 도입하여 이들 메소드에 의해 초래되는 문제를 극복하려고 하였다.

기존의 클래스의 응집도 척도에 대한 연구들은 초기에는 클래스의 고유의 특성을 고려하지 않고 인스턴스 변수와 메소드 간의 관계만을 고려하는 연구로 진행이 되었다. 그러나, 최근에는 위에서 소개된 바와 같이 클래스 고유의 특성을 최대한 응집도 척도에 반영하려는 노력이 진행되고 있다. [5,6,7] 등의 연구는 응집도를 정량화 할 때 특히 클래스의 메소드가 가지는 특성을 반영하는 노력이라고 볼 수 있다. 즉, 메소드에 대한 공용/비공용의 구분, 접근 메소드/생성자/소멸자/위임 메소드의 구분과 같이 메소드의 특성을 척도에 반영하려고 하였다.

본 논문에서는 클래스의 메소드와 마찬가지로 인스턴스 변수에 대해서도 반드시 고려될 특성이 있고 이 특성을 응집도 척도에 반영해야 함을 소개한다. 즉, 인스턴스 변수 중에는 다른 인스턴스 변수에 의해서 값이 결정되는 의존 인스턴스 변수(dependent instance variable)와 그렇지 않은 독립 인스턴스 변수(independent instance variable)가 있으며, 이 두 유형의 인스턴스 변수들은 클래스의 응집도를 측정할 때 구분하여 취급되어야 한다. 기존의 응집도 척도들은 인스턴스 변수 간의 의존성이 척도에 반영되지 않았기 때문에 응집도에 대한 직관적인 평가와는 다른 응집도 값을 낼 수 있는 문제점을 갖게 된다. 본 논문에서는 응집도를 측정할 때 의존 인스턴스 변수의 특성을 특별히 고려해야 할 필요성을 소개하고, 사례 연구를 통하여 의존 인스턴스 변수의 고려 여부가 응집도 측정에 미치는 영향을 보여준다.

본 논문의 구성은 다음과 같다. 2절에서는 클래스의 응집도를 측정하기 위하여 제안된 기존의 대표적인 척도들을 소개한다. 3절에서는 클래스의 인스턴스 변수들을 의존 인스턴스 변수와 독립 인스턴스 변수로 구분하고 인스턴스 변수 간의 의존성이 응집도에 미치는 영향을 설명한다. 4절에서는 사례 연구를 통하여 의존 인스턴스 변수에 대한 고려의 중요성을 설명한다. 마지막으로, 5절에서는 결론과 향후 연구 방향을 기술한다.

2. 기존의 클래스 응집도 척도

객체지향 시스템이 일반화되면서 클래스에 대한 응집도를 측정하기 위한 많은 연구가 수행되었다. 최근에

1) 접근 메소드(access method)는 정보 은닉에 의해서 외부로부터의 직접적인 접근이 불가능한 인스턴스 변수에 대한 조회 및 변경만을 전담하는 메소드이다.

표 1 기존의 클래스 응집도 척도

응집도 척도	설명
LCOM1[8]	공유되는 인스턴스 변수가 없는 메소드 쌍의 수
LCOM2[9]	P를 공유하는 인스턴스 변수가 없는 메소드 쌍의 수이고 Q를 공유하는 인스턴스 변수가 있는 메소드 쌍의 수라고 할 때 LCOM2는 |P| > |Q|이면 |P| - |Q|이고 그렇지 않으면 0이다.
LCOM3[11]	그래프 G에서 클래스의 각 메소드를 노드로 하고 두 메소드가 공유하는 인스턴스 변수가 있으면 해당 노드 사이에 에지를 그릴 때 LCOM3는 그래프 G의 연결 구성 요소(connected component)의 수이다.
LCOM4[11]	LCOM3와 동일하게 계산된다. 단, 호출 관계가 있는 메소드 간에도 그래프 G에 에지를 추가한다.
Co[11]	V를 LCOM4에서 그래프 G의 노드 집합, E를 에지 집합이라고 할 때 $Co = 2.\frac{\|E\|-(\|V\|-1)}{(\|V\|-1)\cdot(\|V\|-2)}$
LCOM5[10]	$\{M_i\}(I \le i \le m)$를 메소드 집합 그리고 $\{A_j\}(1 \le j \le a)$를 인스턴스 변수 집합, $u(A_i)$를 인스턴스 변수 A_j를 참조하는 메소드의 수라고 할 때 $LCOM5 = \frac{(1/a)\sum_{1\le j\le a}\|u(Aj)\|-m}{1-m}$
Coh[6]	LCOM5의 변형으로서, $Coh = \frac{\sum_{1\le j\le a}u(Aj)}{1-m}$
TCC[5]	NP를 공용 메소드 쌍의 수라고 하고, NDC를 직접적으로 연결된 메소드 쌍의 수라고 할 때, TCC = NDC / NP
LCC[5]	NP를 공용 메소드 쌍의 수라고 하고, NIC를 직접적 및 간접적으로 연결된 메소드 쌍의 수라고 할 때, LCC = NIC / NP
CBMC[7]	F_c를 그래프 G의 연결성 요소(connectivity factor), F_s를 구조적 요소(structure factor)라고 할 때, CBMC = $F_c \times F_s$

Briand 등[6]은 대표적인 클래스 응집도 척도를 요약하였다. 표 1은 [6]에서 정리한 응집도 척도를 포함하여 논문[7]에서 제안된 CBMC 척도를 요약하여 보여 준다. 표 1에 기술된 여러 응집도 척도들은 응집도의 측정 기준, 상속 받은 멤버에 대한 처리 방법, 공용/비공용 메소드에 대한 구분, 접근자/생성자/소멸자/위임 메소드에 대한 고려 등에 따라서 분류될 수 있다.

• 응집도 측정 기준에 따른 분류

클래스에 대한 응집도 척도들은 우선 응집도 기준(criteria) 즉, 응집도에 영향을 미치는 요소가 무엇인가에 따라서 세가지로 분류될 수 있다.

• 기준 1: 메소드와 인스턴스 변수간의 관계의 수

이 기준에서는 메소드가 많은 수의 인스턴스 변수를 참조할수록 보다 큰 응집도 값을 갖게 된다. 예를 들어, LCOM[2]5와 Coh 척도가 이 부류에 속한다.

• 기준 2: 공유 인스턴스 변수가 있는 메소드 쌍의 수

이 기준에서는 공통적으로 참조하는 인스턴스 변수를 가지는 메소드 쌍이 많을 수록 큰 응집도를 갖게 된다. 예를 들어, LCOM1, LCOM2, LCOM3, LCOM4, Co, LCC, TCC 척도가 이 부류에 속한다.

• 기준 3: 메소드 간의 연결 강도

최근에 제안된 CBMC[3]에서는 기준 1과 기준 2에 의한 응집도의 정량화 시 발생할 수 있는 문제점을 극복하기 위하여 메소드 간의 연결 강도가 높은 클래스가 높은 응집도를 갖도록 하였으며, 연결성 요소와 구조적 요소는 메소드 간의 연결 강도가 강할 수록 높은 값을 갖도록 정의되었다.

• 상속 받은 멤버에 대한 처리 방법에 따른 분류

클래스는 상위 클래스로부터 인스턴스 변수 및 메소드를 상속 받을 수 있다. 클래스의 응집도를 측정할 때 상속 받은 멤버(인스턴스 변수 및 메소드)에 대한 포함 여부에 따라서 응집도 척도들은 분류 될 수 있다.

• 상속 멤버의 포함

이는 상위 클래스로부터 상속 받은 모든 멤버를 하위 클래스의 응집도를 측정할 때 포함시키는 방법이다. 이 기준은 결국 클래스는 자신이 새롭게 추가한 멤버 뿐만 아니라 상속 받은 멤버를 모두 포함해서 실 세계의 대상을 추상화하므로 상속 받은 멤버도 응집도를 측정할 때 포함시켜야 한다는 관점이다. LCC, TCC, CBMC 척도가 이 부류에 속한다.

• 상속 멤버의 제외

이는 상위 클래스로부터 상속 받은 멤버를 응집도 측정 시 포함하지 않는 방법이다. 이 기준은 하위 클래스에서 새롭게 추가된 멤버들 간의 관련성 정도를 평가하는 측면이 있다. 즉, 기존의 유사한 개념을 상위 클래스로 하여 상속을 받은 후에 하위 클래스에서 정의되는

2) Lack of Cohesion in Methods
3) Cohesion Based onMembe Connectivity

새롭게 추가 또는 변경되는 개념이 자체적으로 얼마나 밀접한 관련성이 있는 가를 측정하려는 의도이다. LCC, TCC, CBMC를 제외한 나머지 모든 응집도 척도들은 이 부류에 속한다.[4)]

- **공용/비공용 메소드의 구분 여부에 따른 분류**

클래스는 정보은닉을 지원한다. 따라서, 메소드라 하더라도 클라이언트에 대한 공개 여부를 지정될 수 있다. 클라이언트 객체로부터 호출될 수 있는 공용 메소드와 그렇지 않은 비공용 메소드[5)]에 대한 구분 여부에 따라서 응집도 척도들을 분류할 수 있다.

• 공용 메소드만을 포함

공용 메소드는 클래스 고유의 기능을 나타내는 반면에, 비공용 메소드는 설계 시 필요에 의해서 정의되는 메소드이다. 즉, 공용 메소드 만이 클래스가 추상화하려는 대상/객체의 행위를 나타내며 비공용 메소드는 시스템 개발 시 설계 단계에서의 판단에 따라서 선택적으로 정의될 수 있는 메소드이므로 필수적인 메소드에 해당되지 않는다. 비공용 메소드가 완전히 제외되는 것은 아니며, 공용 메소드가 비공용 메소드를 통하여 인스턴스 변수를 참고할 때 간접적으로 클래스 멤버 간의 관계를 맺어주는 역할을 하게 된다.

클래스의 구조적인 특성보다는 클래스가 대상/객체를 얼마나 정확하게 추상화 시키고 있는가에 초점을 두는 경우에는 응집도를 측정할 때 공용 메소드만을 포함시킨다. LCC, TCC, CBMC 척도가 이 부류에 속한다.

• 공용/비공용 메소드를 모두 포함

이 경우에는 공용/비공용 메소드를 구분 하지 않고, 모든 메소드를 응집도 측정 시 포함시킨다. 이는 비공용 메소드 또한 클래스를 구성하는 설계적인 요소이므로 전체적인 응집도를 측정할 때는 포함되어야 한다는 관점이다. LCC, TCC, CBMC를 제외한 나머지 응집도 척도들은 공용/비공용 메소드에 대한 구분이 없이 모든 메소드를 응집도를 측정할 때 포함한다.

- **접근 메소드/생성자/소멸자/위임 메소드에 대한 구분 여부에 따른 분류**

접근 메소드는 클라이언트 객체로부터의 접근이 제한된 클래스의 인스턴스 변수에 대한 조회 및 갱신을 지원하는 위한 메소드로서, 정보 은닉의 개념을 지원하기 위하여 클래스가 일반적으로 제공하는 메소드이다. 생성자/소멸자는 객체의 생성/소멸 될 때 호출되는 메소드로서 객체의 초기화/종료 시 수행될 기능을 정의하기 위하여 사용된다. 그리고, 위임 메소드는 클래스가 제공할 기능을 자신의 인스턴스 변수 객체에게 위임(delegation)함으로써 구현하는 메소드이다.

[6,7]에서 언급되었듯이, 접근 메소드/생성자/소멸자/위임 메소드는 클래스의 응집도를 직관적인 판단에 비추어 볼 때 인위적으로 약화시키는 영향을 미친다. 기존의 응집도 척도들은 이와 같은 특별한 메소드에 대한 구분 여부에 따라서 분류될 수 있다.

• 접근 메소드/생성자/소멸자/위임 메소드의 특성 고려

이 방법은 접근 메소드/생성자/소멸자/위임 메소드가 초래하는 문제를 극복하기 위하여 이를 메소드가 미치는 악 영향을 제거하기 위하여 응집도를 측정할 때 이들 메소드를 제외시키는 경우이다. CBMC 척도는 클래스에 정의된 접근 메소드/생성자/소멸자/위임 메소드를 식별하고 응집도를 측정할 때 이 4가지 유형의 메소드를 제외시키고 있으며, LCC와 TCC는 생성자/소멸자를 응집도 측정 시 제외하고 있다.

• 접근 메소드/생성자/소멸자/위임 메소드의 특성 무시

LCC, TCC, CBMC를 제외한 나머지 모든 응집도 척도는 이와 같은 메소드가 클래스의 응집도에 미치는 부정적인 영향을 파악하지 못하며 따라서, 응집도 척도에 이를 메소드의 특성이 반영되지 않고 있다.

3. 의존 인스턴스 변수와 응집도

클래스의 인스턴스 변수는 객체의 상태를 나타내기 위하여 사용되는 변수로서 그 값의 독립성 여부에 따라서 독립(independent) 인스턴스 변수와 의존(dependent) 인스턴스 변수로 구분될 수 있다. 그 값이 다른 인스턴스 변수들의 값에 의해서 결정이 되면 의존 인스턴스 변수이며, 그렇지 않으면 독립 인스턴스 변수이다. 그리고, 의존 인스턴스 변수가 또 다른 의존 인스턴스 변수의 값에 영향을 미칠 수도 있다. 즉, 의존 인스턴스 변수는 다른 독립/의존 인스턴스 변수에 의해서 값이 결정 되는 인스턴스 변수이다.

의존 인스턴스 변수는 그 값이 다른 인스턴스 변수에 의해서 유도될 수 있으므로, 어떤 의미에서는 그 존재가 불필요할 수도 있다. 그러나, 의존 인스턴스 변수는 프로그램에 대한 이해 또는 성능을 높이기 위하여 일반적으로 사용되고 있는 것이 현실이다[13].

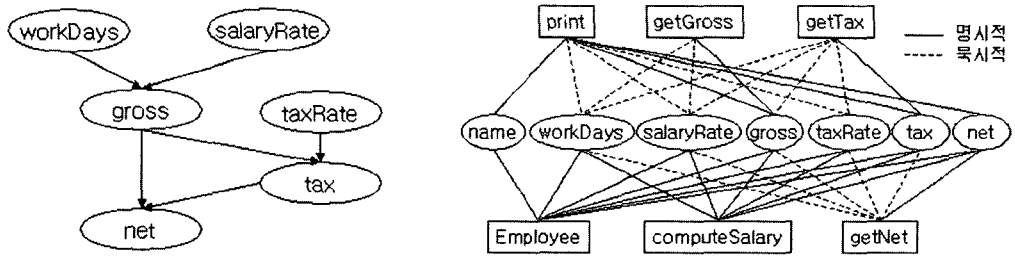
예를 들어, 그림 1의 클래스 *Employee*에서 여러 의존 인스턴스 변수를 발견할 수 있다. 클래스 *Employee*는 7개의 인스턴스 변수 중에서 3개가 의존 인스턴스 변수로 분류될 수 있다. 인스턴스 변수 *gross*는 메소드 *computeSalary()*의 코드에서 파악할 수 있듯이, 인스턴스 변수 *workDays*와 *salaryRate*로부터 값이 결정된다. 마찬가지로, 인스턴스 변수 *tax*와 *net*의 값은 *gross*와 *taxRate* 그리고 *gross*와 *tax*의 값으로부터 각각 유도될 수 있다.

4) 대부분의 응집도 척도들은 상속된 멤버에 대한 처리 여부에 대해서 명확히 언급을 하고 있지 않다.

5) protected 또는 private에 해당

```
class Employee {
    String name ;
    int workDays ;
     float salaryRate, taxRate, net, gross, tax ;
  public:
    Employee(String n, int w, float sr, float tr) :
      name(n), workDays(w), salaryRate(sr), taxRate(tr), net(0), gross(0), tax(0) {;}
    void print() { cout << name << gross << net << tax ; }
    void computeSalary() {
          gross = workdays * salaryRate ;
          tax = gross * taxRate ;
          net = gross - tax ;
    }
    float getGross() { return gross ; }
    float getNet() { return net ; }
    float getTax() { return tax ; }
} ;
```

그림 1 클래스 Employee



(a) 인스턴스 변수간의 의존 관계 (b) 클래스 멤버간의 참조 관계

그림 2 클래스 Employee의 의존 인스턴스 변수

클래스 *Employee*를 구성하는 7 개의 인스턴스 변수 간의 의존 관계를 그래프로 나타내면 그림 2(a)와 같다. 그림 2(a)에서 각 노드는 클래스의 인스턴스 변수를 나타내며 노드 간의 화살표는 해당 인스턴스 변수 간의 의존 관계를 나타낸다. 예를 들어서, 인스턴스 변수 tax는 인스턴스 변수 *gross*와 *taxRate*의 값에 의존하고, 인스턴스 변수 *gross*는 인스턴스 변수 *workDays*와 *salaryRate*에 의존함을 알 수가 있다.

메소드가 *의존* 인스턴스 변수를 접근하는 경우에 이 메소드는 의존 인스턴스 변수를 포함하여 의존 인스턴스 변수의 값에 영향을 미치는 모든 다른 인스턴스 변수와 간접적으로 관련성이 있다고 볼 수 있다. 예를 들어, 메소드 *getGross()*는 의존 인스턴스 변수 *gross*를 참조한다. 따라서, 메소드 *getGross()*는 인스턴스 변수 *gross* 뿐만 아니라, *gross*의 값에 영향을 미치는 인스턴스 변수 *workDays*와 *salaryRate*와도 간접적으로 관련성이 있는 것이다. 다시 말하면, 메소드 *getGross()*가 코드 상에서 명시적으로 *workDays* 및 *salaryRate*를 접근하지는 않고 있지만, 인스턴스 변수들 간의 의존 관계로 인해서 인스턴스 변수 *gross*에 영향을 미치는 모든 인스턴스 변수들과도 관련성이 있는 것으로 판단해야 한다. 따라서, 메소드와 인스턴스 변수 간에는 코드에서 직접 발견할 수 있는 명시적 상호 작용과 인스턴스 변수 간의 의존 관계로부터 유추할 수 있는 묵시적 상호 작용으로도 존재한다.

그림 2(b)는 클래스 Employee의 멤버들 간의 상호작용을 그래프로 표현한 것이다. 그림에서 사각형의 노드는 메소드를 나타내며, 타원의 노드는 인스턴스 변수를 나타낸다. 그리고, 메소드 노드와 인스턴스 변수 노드 사이의 에지는 메소드와 인스턴스 변수 간의 상호 작용으로서 해당 메소드가 인스턴스 변수의 값을 조회하거나 갱신하는 것을 의미한다. 메소드와 인스턴스 변수 간의 실선 에지는 명시적 상호 작용을 나타내며, 점선 에지는 묵시적 상호 작용을 나타낸다. 그림에서 볼 수 있듯이, 클래스 *Employee*에는 메소드와 인스턴스 변수 간의 총 34개의 상호 작용 중에서 20 개의 상호 작용이

메소드가 인스턴스 변수를 직접적으로 참조하는 명시적 상호 작용이며, 나머지 14 개의 상호 작용은 인스턴스 변수 간의 의존 관계에 따른 묵시적 상호 작용이다.

본 논문에서는 명시적 상호 작용 뿐만 아니라, 묵시적 상호 작용도 클래스의 응집도에 영향을 미치므로 응집도의 측정 시 포함되어야 함을 주장한다. 예를 들어, 메소드 *getGross()*는 인스턴스 변수 *gross* 뿐만 아니라, 인스턴스 변수 *workDays*와 *salaryRate*와도 관련성이 있는 것이므로 메소드 *getGross()*와 인스턴스 변수 *workDays* 및 *salaryRate*와의 관계도 응집도를 측정할 때 반영되어야 한다.

기존의 모든 응집도 척도들은 메소드와 인스턴스 변수 간의 명시적 상호 작용만을 고려하였으며 인스턴스 변수 간의 의존 관계에 따른 묵시적 상호 작용은 응집도를 측정할 때 반영되고 있지 않았다. 따라서, 실제로 존재하는 멤버 간의 관계가 응집도 측정 시 누락되므로 직관적인 응집도에 비하여 보다 낮은 응집도 측정 값을 산출하는 문제점을 초래한다.

LCOM5, Coh 척도와 같이 메소드가 참조하는 인스턴스 변수의 수에 따라서 응집도를 결정하는 경우에는 실제 응집도에 기여하는 묵시적인 상호 작용이 포함되지 않으므로 낮은 값을 산출하게 된다. 마찬가지로, 그 외의 응집도 척도에서는 공유하는 인스턴스 변수를 가지는 메소드 쌍을 그렇지 않은 메소드 쌍으로 간주함으로 인해서 낮은 응집도 값을 초래할 수 있다. 예를 들어, 기존의 응집도 척도에서는 메소드 *getTax()*와 *getNet()*는 각각 *tax*와 *net*을 참조하므로 공유하는 인스턴스 변수가 없는 메소드 쌍이므로 클래스의 응집도를 낮추는 영향을 미친다. 그러나, 그림 2(b)에서 볼 수 있듯이, 두 메소드는 명시적으로는 공유하는 인스턴스 변수가 존재하지는 않지만, 인스턴스 변수들 간의 의존 관계로 인해서 인스턴스 변수 *tax, taxRate, gross, salaryRate, workDays*를 간접적으로 공유하고 있다고 볼 수 있다.

예를 들어, 표 2는 기존의 응집도 척도에 대해서 의존 인스턴스 변수에 의한 묵시적인 상호 작용을 포함시키기 전과 후에 클래스 *Employee*의 응집도를 측정한 값을 보여 준다.

- LCOM1, Co, LCOM5, Coh, TCC 척도의 경우에는 기존의 측정에 비하여 보다 향상된 응집도 값을 산출함을 알 수가 있다. 이는 클래스 *Employee*에 존재하는 의존 인스턴스 변수에 의한 묵시적 상호 작용이 반영되었기 때문이다. Co, Coh, TCC 척도는 측정 값이 클수록 강한 응집도를 나타내고, LCOM1, LCOM5는 측정 값이 작을수록 강한 응집도를 나타낸다.
- LCOM3와 LCOM4 척도에 대해서는 묵시적 상호 작용의 고려 여부와 관계 없이 동일한 측정 값을 보이고 있다. 이는 LCOM3와 LCOM4 척도가 정의될 때 의도적으로 의존 인스턴스 변수의 영향을 고려한 것은 아니지만, 척도의 정의에 의해서 묵시적인 상호작용이 반영되고 있기 때문이다.LCOM3와 LCOM4는 클래스의 멤버들 간의 참조관계를 그래프로 표현하였을 때의 연결 요소(connected component)의 수로 정의된다. 의존 관계가 있는 두 인스턴스 변수는 두 인스턴스 변수를 모두 접근하는 한 개 이상의 메소드가 있는 것을 의미한다. 따라서, 의존 관계의 고려 여부에 관계 없이 이미 클래스의 멤버들은 두 인스턴스 변수를 모두 참조하는 메소드에 의해서 연결되어 있다. 그러므로, 연결 요소의 수로 정의되는 LCOM3와 LCOM4 척도의 경우에는 의존 인스턴스 변수에 대한 고려 여부와 관계 없이 동일한 측정 값을 산출하게 된다. 그러나, 이는 LCOM3와 LCOM4가 의존 인스턴스 변수의 특성을 반영하려고 했던 것은 아니며, 척도의 정의에 따른 의도하지 않은 결과일 뿐이다.
- LCOM2, LCC, CBMC 척도에 대해서는 묵시적 상호 작용의 고려 여부와 관계 없이 동일한 측정 값을 보이고 있다. 그러나, 클래스 *Employee*의 경우에는 묵시적 상호 작용이 LCOM2, LCC, CBMC의 값에는 영향을 미치지 않기 때문이며, LCOM3와 LCOM4와 같이 항상 의존 인스턴스의 고려 여부와 관계 없이 항상 동일한 응집도 값을 산출하는 것은 아니다. 그리고, 4절의 사례 연구에서는 의존 인스턴스 변수의 특성에 대한 고려가 LCOM2, LCC, CBMC의 값에 미치는 영향을 보여 준다.

표 2 의존 인스턴스 변수 고려 여부에 따른 클래스 *Employee*의 응집도 측정 값 비교

이름	고려 전 측정 값	고려 후 측정 값
LCOM1	3	0
LCOM2	0	0
LCOM3	1	1
LCOM4	1	1
Co	0.70	1
LCOM5	0.63	0.23
Coh	0.48	0.81
TCC	0.70	1
LCC	1	1
CMBC	0.50	0.50

4. 사례 연구

이 절에서는 본 논문에서 소개한 의존 인스턴스 변수에 대한 고려의 중요성을 보여 주기 위하여 수행된 사례 연구를 소개한다.

4.1 응집도 측정 도구

우리는 표 1에 소개된 응집도 척도들을 자동적으로 계산할 수 있는 응집도 측정 도구를 개발하였다. 이 응집도 측정 도구는 기존의 척도 뿐만 아니라, 데이타 흐름 분석 기법을 이용하여 의존 인스턴스 변수의 영향을 반영할 수 있도록 수정된 척도에 대해서도 응집도 값을 계산할 수 있다. 응집도 측정 도구는 GEN++[14]를 이용하여 C++ 프로그램으로부터 응집도 측정에 필요한 정보를 추출하고 이를 바탕으로 클래스에 대한 응집도를 자동으로 계산할 수 있다. 그림 3은 응집도 측정 도구가 GEN++ 도구를 이용하여 C++ 소스 코드로부터 클래스에 대한 정보를 추출하고, 클래스에 대한 브라우저와 응집도 측정기능을 제공함을 나타낸다.

그림 4는 개발된 응집도 측정 도구가 제공하는 클래스 브라우징 기능과 응집도 측정 기능의 사용에 대한 예시 화면이다. 그림 4(a)는 클래스 브라우저 창으로서 입력 C++ 프로그램에 정의된 각 클래스 별로 상위 클래스, 인스턴스 변수 및 메소드와 정의된 소스 코드를 보여 준다. 그리고, 그림 4(b)는 응집도 측정 결과를 보여 준다.

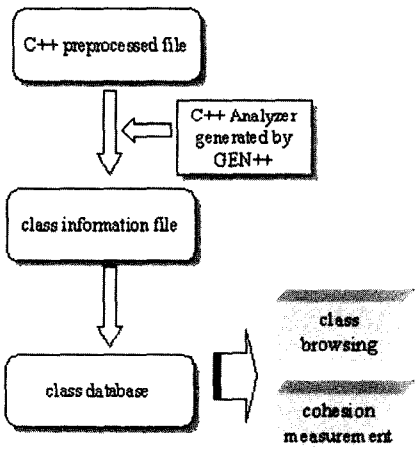


그림 3 응집도 측정 도구의 구조

4.2 인스턴스 변수 간의 의존 관계 파악

인스턴스 변수 간의 의존 관계를 파악하기 위해서는 클래스의 각 메소드의 구현 코드를 분석해야 한다. 이 절에서는 인스턴스 변수들 간의 의존 관계를 파악하는 방법을 기술한다.

한 인스턴스 변수의 값이 다른 인스턴스 변수의 값으로부터 계산될 때 두 인스턴스 변수간에는 의존 관계가 성립된다. 의존 관계는 한 메소드 내부의 문장에서 발생할 수도 있으며, 메소드 간의 호출에 의해서도 발생할 수가 있다.

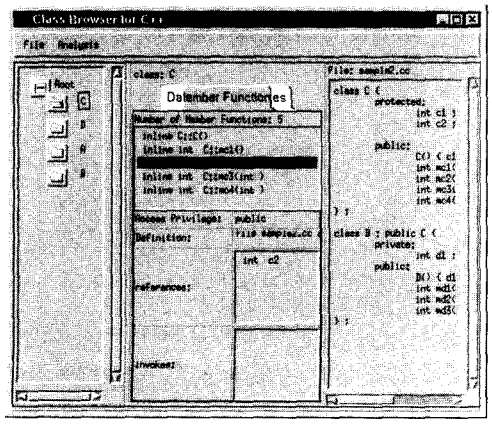
4.2.1 메소드 내부의 의존 관계

정의 1. 메소드의 한 문장 s에 의한 변수 간의 의존 관계는 $VD_S(s)$로 표현되며 다음과 같이 정의 된다.

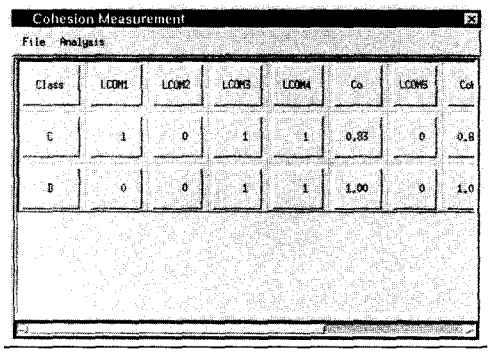
$VD_S(s)=\{<i,j> \mid i$는 s에서 정의된 변수이고 j는 i를 정의하는데 사용된 변수이다.$\}$

예를 들어, 그림 5는 클래스 A의 코드를 보여 준다. 클래스 A의 메소드 Af1()은 4개의 대입문을 가지고 있으며, 각 대입문에 의해서 변수들 간의 의존 관계가 발생한다. 예를 들어, s1문장에 의한 의존 관계는 $VD_S(s1)$ = {<v, av1>}, $VD_S(s2)$ = {<av2, v>}, $VD_S(s3)$ = {<av3, v>, <av3, x>}, $VD_S(s4)$ = {<av4, av2>}이다.

변수 간의 의존 관계는 문장의 수행 경로에 따라서 달라질 수 있다. 따라서, 메소드 내의 각 수행 경로 별로 의존 관계가 정의될 수 있다.



(a) 클래스 브라우저 창



(b) 응집도 측정 창

그림 4 응집도 측정 도구의 사용 예시 화면

```
class A {
    int av1, av2, av3, av4 ;
  public:
    void Af1(int x) {
    s1 : int v = av1 ;
    s2 : if ( av1 > 0 ) av2 = v ;
    s3 : else av3 = v + x ;
    s4 : av4 = av2 ;
    }
    void Af2() { av1 = av2 ; }
} ;
```

그림 5 클래스 A

정의 2. p_n=<s_1, s_2, …, s_n>를 n개의 문장으로 구성된 경로라고 할 때, 경로 p_n(=p)에 따른 의존 관계는 $VD_P(p_n)$로 표현되며 다음과 같이 정의된다.

$$VD_P(p_n)=\begin{cases} VD_S(s_n) & ,n=1 \\ VD_P(p_{n-1}) \cup VD_S(s_n) & n \geq 2 \end{cases}$$

그리고, 의존 관계는 이행적인 관계이다. $VD^*_P(p)$를 $VD_P(p)$의 이행적 폐쇄(transitive closure)라고 정의한다. 예를 들어, 메소드 Af1()은 두 개의 수행 경로 p1=<s1, s2, s4>와 p2=<s1, s3, s4>를 가지고 있으며, p1경로에 대해서는 $VD^*_P(p1)$={<v, av1>, <av2, v>,<av2, av1>, <av4, av2>, <av4, v>, <av4, av1>}가 되며, p2경로에 대해서는 $VD^*_P(p2)$={<v, av1>, <av3, v>, <av3, x>, <av3, av1>, <av4, av2>}가 된다.

그리고, 한 메소드 내에서의 의존 관계는 메소드에 포함된 각 경로 별 의존 관계를 조합함으로써 결정된다.

정의 3. $P(m)$를 메소드 m의 모든 수행 경로의 집합이라고 할 때, 메소드 m에 의한 의존 관계는 VD_M(m)로 표현되며 다음과 같이 정의된다.

$$VD_M(m)=\bigcup_{p \in P(m)} VD^*_P(p)$$

예를 들어, 클래스 A의 메소드 Af1()에 의한 의존 관계는 VD_M(Af1()) = VD^*_P(p1) U VD^*_P(p2)이다.

4.2.2 메소드 간의 의존 관계

변수 간의 의존 관계는 한 메소드 내의 수행 경로 뿐만 아니라, 메소드 간의 호출 관계에 의해서도 발생할 수 있다. 예를 들어, 그림 6에서 클래스 C의 메소드 Cf1()의 문장 s1에서 클래스 B의 객체 b의 메소드 Bf1()을 호출함으로써 인스턴스 변수 cv1에 의해서 인스턴스 변수 b의 값이 변경될 수 있다. 따라서, b는 cv1에 의존한다고 판단될 수 있다. 본 논문에서는 이와 같이 메소드 간의 호출에 따른 의존 관계를 세 가지로 분류하여 살펴 본다.

문장 s가 r=t.cm(a1, a2, …, an)이고, 객체 t는 클래스 T의 인스턴스이며, cm(f1, f2, …, fn)를 피호출 메소드 cm의 원형(prototype)이라고 하자. 그리고, R(cm)을 메소드 cm의 **return** 문에 나타나는 변수들이라고 하고, R'(cm)을 메소드 cm의 **return** 값에 영향을 주는 변수들이라고 한다. 즉, R'(cm) = R(cm) U {v | ∃w∈R(cm) • <v, w>∈VD_M(cm)}이다.

- 실 매개변수와 대상 객체 사이의 의존 관계

  • 대상 객체가 실 매개변수에 의존하는 경우

실 매개 변수 ai에 대응되는 형식 매개변수 fi가 메소드 m에서 객체 t의 인스턴스 변수의 값을 변경하는 경우, t의 ai에 대한 의존 관계를 추가한다.

∃v∈V(T) • <v, fi>∈VD_M(cm)→VD_S(s) = VD_S(s) U {<t, ai>}

예를 들어, 그림 6에서 클래스 C의 메소드 Cf1()의 문장 s1을 보면 실 매개변수 cv1의 값이 피호출 메소드 B::Bf1()에서 객체 b의 인스턴스 변수 bv의 값에 영향을 준다. 따라서, 인스턴스 변수 b는 인스턴스 변수 cv1에 의존한다. 즉, VD_S(s1)={<b, cv1>}이다.

• 실 매개변수가 대상 객체에 의존하는 경우

실 매개변수 ai가 피호출 메소드 cm에서 값이 정의되는 경우, ai의 t에 대한 의존 관계를 추가한다.

∃v∈V(T) • <ai, v>∈VD_M(cm)→VD_S(s) = VD_S(s) U {<ai, t>}

예를 들어, 메소드 Cf1()의 문장 s2에서 실 매개변수 cv2는 피호출 메소드 B::Bf2()에서 값이 변경된다. 따라

```
class B {
    int bv ;
  public:
    void Bf1(int x) { bv = x ; }
    void Bf2(int* x) { *x = bv ; }
    int Bf3(int x) { return x +bv ; }
} ;
```

```
class C {
    int cv0, cv1, cv2 ;
    B b ;
  public:
    void Cf() {
      s1 : b.Bf1(cv1) ;
      s2 : b.Bf2(&cv2) ;
      s3 : cv1 = b.Bf3(cv0) ;
    }
} ;
```

그림 6 클래스 B와 C

서, cv2는 b에 의존하게 된다. 따라서, $VD_S(s2)$={<cv2, b>}가 된다.

• 메소드의 반환 값과 대상 객체 사이의 의존 관계

만약 객체 t의 인스턴스 변수들이 피호출 메소드 cm의 **return** 값에 영향을 주고, 그 **return** 값이 r에 대입된다면, r은 t에 의존하게 된다.

$\exists v \in V(T),\ w \in R'(cm) \cdot <v,\ w> \in VD_M(cm) \rightarrow VD_S(s) = VD_S(s) \cup \{<r, t>\}$

예를 들어, 메소드 Cf1()의 문장 s3에서 객체 b의 인스턴스 변수 bv가 메소드 Bf3()의 **return** 값에 영향을 주고, 이는 대입문에 의해서 cv1에 영향을 준다. 따라서, $VD_S(s3)$={<cv1, b>}이다.

• 메소드의 반환 값과 실 매개 변수 사이의 의존 관계

만약 실 매개변수 ai에 대응되는 형식 매개변수 fi가 피호출 메소드 cm의 **return** 값에 영향을 주고, 메소드 m의 호출의 **return** 값이 r에 대입된다면, r은 ai에 의존하게 된다.

$\exists w \in R'(cm) \cdot <fi,\ w> \in VD_M(cm) \rightarrow VD_S(s) = VD_S(s) \cup \{<r, ai>\}$

예를 들어, 메소드 Cf2()의 문장 s3에서 인스턴스 변수 cv0는 메소드 Bf3()의 **return** 값에 영향을 주고, 이는 대입문에 의해서 cv1에 영향을 준다. 따라서, $VD_S(s3)$={<cv1, b>, <cv1, cv0>}이다.

4.2.3 인스턴스 변수 간의 의존 관계

클래스의 인스턴스 변수간의 의존 관계는 각 메소드에 의한 인스턴스 변수 간의 의존 관계를 조합함으로써 결정된다. 그리고, 의존 관계는 다음의 특성을 만족해야 한다.

- 의존 관계는 비반사적(irreflexive)이다. 즉, 인스턴스 변수는 자기 자신의 값에 의존할 수 없다.
- 의존 관계는 비대칭적(asymmetric)이다. 즉, 인스턴스 변수 v가 w에 의존한다면, 반대로 w는 v에 의존할 수 없다.
- 의존 관계는 이행적(transitive)이다. 즉, 인스턴스 변수 v가 w에 의존하고, w는 x에 의존한다면, v는 x에도 의존한다.

정의 4. V(c)와 M(c)를 클래스 C의 인스턴스 변수의 집합과 메소드 집합이라고 할 때, 클래스 C의 인스턴스 변수간의 의존 관계는 IVD(c)로 표현되며 다음과 같이 정의된다.

$IVD(c)=\{<vi, vj> \mid <vi, vj> \in VD_C(c) \wedge i \neq j \wedge <vj, vi> \notin VD_C(c) \wedge vi \in V(c) \wedge vj \in V(c)\}$,

$$VD_C(c) = \bigcup_{m \in M(c)} VD_M(m)$$

예를 들어, 클래스 C의 인스턴스 변수들 간의 의존 관계는 IVD(c) = {<cv1, cv0>, <cv2, b>, <cv2, cv1>, <cv2, cv0>}이다.

4.3 실험 결과

이 논문에서는 인스턴스 변수의 의존 관계에 대한 고려 여부가 클래스 응집도에 미치는 영향을 파악하기 위하여 응집도 측정 도구를 이용하여 InterViews를 대상으로 실험을 수행하였다. InterViews는 Stanford에서 개발된 C++ 클래스 라이브러리로서 X 윈도우 상에서의 GUI를 제공하기 위한 많은 클래스로 구성되어 있다.

우리는 이 실험에서 InterViews의 102개의 클래스에 대해서 인스턴스 변수간의 의존 관계에 대한 고려가 없는 기존의 응집도 척도를 적용한 경우와 의존 관계를 고려하여 응집도 척도를 개선하였을 경우의 결과를 비교하였다.

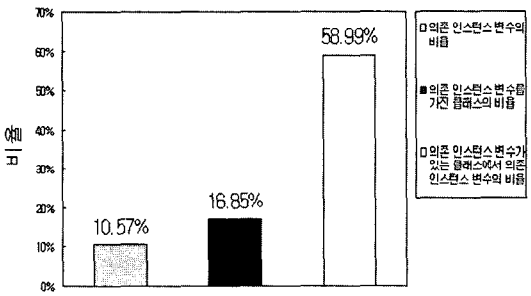
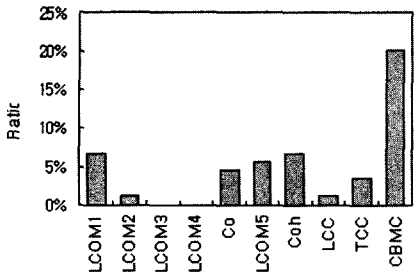


그림 7 InterViews 시스템에서 발견된 의존 인스턴스 변수

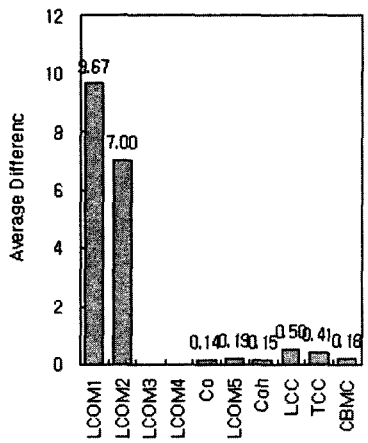
그림 7은 InterViews 시스템에서 발견된 의존 인스턴스 변수의 비율을 보여 준다. 첫 번째는 모든 클래스의 총 인스턴스 변수 수에 대한 의존 인스턴스 변수 수의 비율로서 약 11% 정도이다. 두 번째는 전체 클래스의 수에 대해서 1개 이상의 의존 인스턴스 변수를 가지고 있는 클래스의 수에 대한 비율로서 약 17% 정도이다. 그리고, 세 번째는 의존 인스턴스 변수를 가진 클래스에 대해서 클래스의 전체 인스턴스 변수의 수에 대한 의존 인스턴스 변수의 비율로서 약 59% 정도이다.

이 그림은 InterViews라는 하나의 시스템에 대한 측정이므로 이 결과를 일반적인 현상이라고 주장하기에는 부족한 데이타 일 수는 있지만, 이 결과는 [13]에서 언급된 것처럼 개발자들은 사용의 편이성 또는 성능 상의 이유로 의존 인스턴스 변수가 적지 않게 사용됨을 보여준다. 따라서, 인스턴스 변수 간의 의존 관계가 응집도 측정 시 반영될 필요성이 있음을 확인할 수 있었다.

그림 8은 InterViews 시스템의 각 클래스에 대해서 의존 관계의 고려 여부가 응집도 측정에 미치는 영향을 보여 준다. 이 그림을 통하여 실제 시스템에서 인스턴스 변수간의 의존성이 존재하며 이는 클래스의 응집도 측



(a) 의존 인스턴스 변수의 영향



(b) 기존 응집도 값과의 차이

그림 8 의존 인스턴스 변수가 응집도에 미치는 영향

정에 영향을 미치고 있음을 확인할 수 있다.

- 그림 8(a)는 의존 인스턴스 변수를 고려하도록 기존의 응집도 척도의 정의를 수정한 후에 측정하였을 때 응집도의 변화가 있는 클래스의 수의 비율을 보여 준다. LCOM3와 LCOM4를 제외한 나머지 척도들에서는 응집도에 변화가 관찰된 클래스가 발견되었다. LCOM3와 LCOM4 척도는 3절에서 설명한 것처럼 의존 인스턴스 변수의 영향을 의도한 것은 아니지만 척도의 정의상 인스턴스 변수 간의 의존 관계가 척도에 영향을 미치지 않기 때문이다. 다른 응집도 척도에 비하여 CBMC에서 의존 인스턴스 변수에 영향을 받는 클래스가 많은 이유는 CBMC는 멤버들 간의 연결성에 의해서 결정되며 의존 인스턴스 변수에 따른 묵시적인 상호작용에 의해서 기존에는 발견하지 못하였던 멤버들 간의 관계가 이제 고려가 되었기 때문이다.
- 그림 8(b)는 의존 인스턴스 변수에 의해서 응집도 값이 달라진 클래스에 대해서 원래의 응집도 값과의 차이의 평균 값을 보여 준다. LCOM1과 LCOM2가 다른 척도에 비하여 측정된 응집도가 큰 차이를 보이는 것은 LCOM1과 LCOM2 이외의 척도들은 응집도 값이 0과1 사이(LCOM5는 0과 2사이)로 정규화가 되어 있지만, LCOM1과 LCOM2 척도는 최대 값이 정해져 있지 않기 때문이다.

5. 결론 및 향후 연구 방향

응집도는 고 품질 설계의 판단 기준의 하나로서 일반적으로 많이 사용되고 있다. 따라서, 클래스의 응집도를 정확하게 평가하는 것은 매우 중요한 일이다. 기존의 응집도 척도들은 의존 인스턴스 변수라는 클래스의 특성을 반영하고 있지 않기 때문에 클래스의 응집도에 대한 정확한 평가가 이루어지지 않을 수가 있다.

본 논문에서는 클래스에 존재하는 의존 인스턴스 변수의 영향을 살펴 보고, 이를 반영하여 기존의 응집도 척도를 향상 시킬 필요가 있음을 설명하였다. 특히, C++ 클래스 라이브러리에 대한 사례 연구를 통하여 의존 인스턴스 변수의 실제 사용 빈도와 이에 따른 응집도 측정 값에 미치는 영향을 실험을 통하여 조사하였다.

앞으로 개선된 응집도 척도들이 기존의 응집도 척도보다 클래스의 품질을 좀 더 정확하게 반영할 수가 있는 가에 대한 실험적인 연구가 필요하다. 즉, 클래스의 유지보수도, 이해도, 재사용도 등에 대한 자료와 실제 응집도의 측정 값과의 상관 관계를 조사함으로써 의존 인스턴스 변수에 대한 고려의 중요성을 실험적으로 확인할 수 있을 것이다.

참 고 문 헌

[1] W. Stevens, G. Myers and L. Constantine, "Structured Design," IBM Systems Journal, Vol, 12, No. 2, 1974.

[2] N. N. Card, G. T. Page and F. E. McGarry, "Criteria for Software Modularization," Proc. of 8th Int. Conf. on Software Engineering, pp. 372-377, 1985.

[3] N. N. Card, V. E. Chruch and W. W. Agresti, "An Empirical Study of Software Design Practices," IEEE Trans. on Software Engineering, Vol 12, No. 2, pp. 264-271, 1986.

[4] A. Snyder, "Encapsulation and Inheritance in Object-Oriented Programming Languages," Proc. of 1th ACM Conf. on Object-Oriented Systems, Languages, and Applications, pp. 84-91, September 1986.

[5] J. M. Bieman and B. -K. Kang, "Cohesion and Reuse in an Object-Oriented System," in: Proc. Symp. on Software Reusability, pp. 259-262, 1995.

[6] L. C. Briand, J. . Daly and J. Wust, "A Unified Framework for Cohesion Measurement in Object-Oriented Systems," Empirical Software Engineering Journal, Vol. 1, pp. 65-117, 1998.

[7] H. -S. Chae, Y. -R. Kwon and D. -H. Bae, "A

Cohesion Measure for Object-Oriented Classes," Software Practice and Experience, Vol. 30, pp. 1405-1431, 2000.
[8] S. R. Chidamber and C. F. Kemerer, "Towards a Metrics Suite for Object-Oriented Design," in: Proc. 6th ACM Conf. on Object-Oriented Systems, Languages and Applications, pp. 197-211, 1991.
[9] S. R. Chidamber and C. F. Kemerer, "A Metrics Suite for Object-Oriented Design," IEEE Trans. on Software Engineering, Vol. 20, pp. 476-493, 1994.
[10] B. Henderson-Sellers, Software Metrics, Prentice-Hall, 1996.
[11] M. Hitz and B. Montazeri, "Measuring Coupling and Cohesion in Object-Oriented Systems," in: Proc. of Symp. on Applied Corporate Computing, 1995.
[12] W. Li and S. Henry, "Object-Oriented Metrics that Predict Maintainability," Journal of Systems and Software, Vol. 23, pp. 111-122, 1993.
[13] James Rumbaugh, Michael Blaha, William Premerlani, Frederick Eddy, and William Lorensen, Object-Oriented Modeling and Design, Prentice Hall International, 1991.
[14] P. Devanbu, "GENOA a customizable, language- and front-end independent code analyzer," in: Proc. Conf. on Software Engineering, 1992, 307-317.

채 홍 석

1994년 서울대 원자핵공학 학사. 1996년 한국과학기술원 전산학 석사. 2000년 한국과학기술원 전산학 박사. 2000년~2003년 ㈜ 동양시스템즈 기술연구소 선임연구원. 2003년~2004년 한국과학기술원 전산학과 초빙교수. 2004년~현재 부산대학교 컴퓨터 공학과 전임강사. 관심분야는 객체지향 방법론, 소프트웨어 테스팅, 소프트웨어 메트릭, 소프트웨어 유지보수

권 용 래

1969년 서울대학교 문리대학 이학사
1971년 서울대학교 대학원 이학석사
1971년~1974년 육군사관학교 전임강사
1978년 미국 피츠버그대학 이학박사
1978년~1983년 미국 Computer Science Corporation 연구원. 1983년~현재 한국과학기술원 전자전산학과 전산학전공 교수. 2002년 한국정보과학회 수석부회장. 2003년 한국정보과학회 회장. 관심분야는 실시간 병렬 소프트웨어 검증, 실시간 시스템의 객체지향 기술, 고신뢰도 소프트웨어의 품질 보증, 소프트웨어 시험 기법

배 두 환

1980년 서울대학교 조선공학과 학사. 1987년 Univ. of Winsconsin - Milwaukee, 전산학 석사. 1992년 Univ. of Florida, 전산학 박사. 1992년~1994년 University of Florida 전산학과 교수. 1995년~현재 한국과학기술원 전자전산학과 교수. 관심분야는 소프트웨어 프로세스, 컴포넌트 기반 소프트웨어 공학, 객체지향기술