클래스 멤버 사이의 쓰기 연산을 고려한 새로운 TCC 및 LCOM 척도

(New TCC and LCOM Measures Considering the Write Operations between Class Members)

우 균 † 채 흥 석 ††
(Gyun Woo) (Heung Seok Chae)

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.

본 논문에서는 클래스 멤버 간의 쓰기 의존성의 영향을 고려함으로써 기존의 응집도 척도를 개선시킬 수 있음을 소개한다. 본 논문의 기법을 지원하는 도구를 개발하였으며 이 도구를 이용하여 C++ 클래스 라이브러리에 이 기법을 적용한 사례 결과를 소개한다.

키워드 : 객체지향 시스템, 클래스 응집도, 쓰기 의존성

Abstract Cohesion refers to the degree of the relatedness of the members in a class and it is widely accepted that the higher the cohesion of a module is, the easier the module to understand and maintain. Recently, several cohesion measures have been proposed to measure the cohesiveness of classes in an object-oriented program. In this paper, we propose an approach to improving the existing cohesion measures by considering the impact of write dependencies between class members. We have developed a cohesion measurement tool for supporting our approach and describe a case study performed with a C++ class library.

Key words : object-oriented systems, class cohesion, write dependency

1. 서 론

응집도는 구조적 설계 기법에서 좋은 설계를 판단하는 기준으로서 도입되었으며, 모듈의 구성 요소들 사이의 관련성 정도를 나타내는 척도이다[1]. 높은 응집도의 모듈은 오직 하나의 기능 만을 제공하며 모듈의 모든 구성 요소들이 그 기능을 구현하기 위하여 서로 밀접하게 상호작용을 한다. 반면에, 낮은 응집도의 모듈은 관련성이 적은 구성 요소들로 구성이 되고, 여러 기능을 제공하는 것이 일반적이다. 그리고 응집도가 높은 모듈은 개발, 유지보수, 재사용이 용이하다고 알려져 있다[2,3]. 따라서, 시스템 설계 시 가능한 한 높은 응집도의 모듈로서 설계될 수 있도록 하며, 유지 보수 단계에서도 높은 수준의 응집도를 유지할 수 있도록 노력하는 것이 바람직하다. 지금까지 모듈의 응집도 척도들이 많이 제안되었으며 이들 척도들은 개발자들이 높은 응집도의 모듈을 개발할 수 있도록 모듈의 응집도를 평가할 때 사용되고 있다.

객체지향 시스템은 실 세계에 존재하는 물리적인 개체 및 추상적인 개념을 객체/클래스로 모델링함으로써 기존의 개발 방법에 비하여 보다 높은 생산성과 유지보수성을 제공하는 것으로 인정되고 있다. 클래스는 실 세계의 중요한 대상을 추상화시킨 모델로서 객체지향 시스템의 개발에서 가장 중요한 역할을 한다. 즉, 실 세계에 존재하는 중요한 대상은 클래스로서 모델링되고, 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, 클래스

· 본 연구는 정보통신부 및 정보통신연구진흥원의 대학 IT연구센터 육성·지원사업의 연구결과로 수행되었음
† 종신회원 : 부산대학교 컴퓨터공학과 교수
woogyun@pusan.ac.kr
†† 정 회 원 : 부산대학교 컴퓨터공학과 교수
hschae@pusan.ac.kr
논문접수 : 2005년 1월 31일
심사완료 : 2005년 9월 14일

의 인스턴스(객체)는 반드시 클래스에 정의된 인터페이스를 통해서만 조작될 수 있다. 따라서, 기존의 인터페이스를 만족시키는 범위내에서 클래스의 내부(인스턴스 변수, 메소드의 구현)의 변경은 시스템의 다른 부분에 전혀 영향을 미치지 않으므로 시스템의 유지보수성 및 확장성에 큰 기여를 하게 된다[4].

객체지향 시스템의 이점에 대한 성취 여부는 클래스의 품질에 의존한다. 따라서 객체지향 방법론에서는 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, 단순히 별 관련성이 없는 멤버들의 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 기존의 구조적 설계 기법에서의 모듈의 개념에 따라서 응집도를 클래스를 구성하는 요소 간의 참조(reference) 여부와 같은 단순한 관련성 정도만을 바탕으로 정의되었으며, 클래스 고유의 특성을 반영하지 못하였다. 그리고, 이처럼 기존의 모듈과는 다른 클래스 고유의 특성을 반영하지 못하는 문제점을 가지고 있음이 여러 연구에서 지적되었다[5-7, 13].

예를 들어, Bieman 등은 공용 메소드와 비공용 메소드를 구분하고, 공용 메소드들만의 관련성 정도로서 LCC(Loose Class Cohesion)과 TCC(Tight Class Cohesion) 척도를 제안하였다[5]. 이는 비공용 메소드[1]도 클래스의 구성 요소이지만 비공용 메소드는 공용 메소드와 달리 다른 객체에 의해서 직접적으로 이용이 되지 않으며 공용 메소드의 구현을 돕기 위한 역할을 할 뿐이기 때문이다. 즉, 비공용 메소드의 사용 여부는 클래스가 제공할 기능/서비스에 무관하며 클래스에 대한 설계의 결과(design decision)에 따라서 그 존재여부가 달라지기 때문이다. 또한, Bieman 등은 객체지향 프로그램에서 빈번히 사용되는 생성자와 소멸자에 의해서 발생할 수 있는 문제점을 해결하기 위하여 응집도의 측정시 생성자와 소멸자를 제외하였다. Briand 등은 접근 메소드[2]가 클래스의 응집도를 불필요하게 약화시키는 문제점을 지적하고 이들 접근 메소드를 제외하고 응집도를 측정해야 한다고 주장하였다[6]. 또한, CBMC (Cohesion Based on Member Connectivity)[7]에서는 접근 메소드, 생성자, 소멸자와 더불어 위임 메소드도 동일한 문제를 유발할 수 있음을 지적하고, 이를 특별 메소드라는 개념을 도입하여 이들 메소드에 의한 문제를 극복하려고 하였다. 또한, 클래스의 인스턴스 변수와 메소드간의 관계 뿐만 아니라, 인스턴스 변수와 인스턴스 변수 간의 의존 관계를 추가적으로 반영하는 연구도 있다[14]. 최근에는 클래스의 인스턴스 변수 간의 의존 관계에 의한 메소드 간의 간접적인 상호작용을 포함해야 한다는 연구도 진행되었다[13].

요약하면, 기존의 클래스의 응집도 척도에 대한 연구들은 초기에는 클래스의 고유의 특성을 고려하지 않고 인스턴스 변수와 메소드 간의 관계만을 고려하는 연구로 진행이 되었으나, 최근에는 위에서 소개된 바와 같이 클래스 고유의 특성을 최대한 응집도 척도에 반영하려는 노력이 진행되고 있다. 특히 클래스의 메소드 및 인스턴스 변수가 가지는 특성을 응집도 정량화 과정에 반영하려는 연구가 진행된 바 있다. 즉 이 연구들은 메소드에 대한 공용/비공용의 구분, 접근 메소드/생성자/소멸자/위임 메소드의 구분, 의존 변수에 의한 간접적인 상호작용 등을 고려함으로써 클래스의 특성을 응집도 척도에 반영하고자 하였다[5-7,13-15].

본 논문에서는 클래스 멤버 간의 쓰기 의존성의 영향을 고려함으로써 기존의 응집도 척도를 개선시킬 수 있는 방법을 소개한다. 일반적으로 클래스의 응집도는 클래스 멤버 간의 상호작용의 정도에 의해서 결정된다. 그리고, 멤버 간의 상호 작용은 읽기 상호작용과 쓰기 상호작용으로 분류될 수 있다. 기존의 응집도 척도들은 두 가지 유형의 상호작용을 구분하지 않고 동일한 역할을 하는 것으로 간주하였다.

그러나 메소드가 한 인스턴스 변수에 쓰기 연산을 수행하였다면 그 인스턴스 변수를 읽는 다른 모든 메소드에 영향을 미칠 수 있기 때문에 쓰기 연산과 읽기 연산을 구분하여 응집도를 정의하는 것이 더 합리적이라고 생각한다. 본 논문에서는 읽기 연산에 비하여 쓰기 연산이 클래스의 응집도에 보다 큰 영향을 미치는 것으로

1) Java 언어와 C++ 언어에서 protected와 private 멤버에 해당한다.

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

간주한다.

본 논문에서는 이와 같은 쓰기 상호작용의 특성을 고려하여 기존의 응집도 척도를 개선하는 기법을 소개한다. 구체적으로 말하면, 클래스 응집도 척도 중에서 TCC[5]와 LCOM[9]을 대상으로 쓰기 의존성을 반영한 개선 방법을 설명한다. 이 두 개의 응집도 척도는 비교적 많은 실험적인 연구[16-19]에서 사용이 되고 있으며 여러 CASE 도구에서도 지원되고 있기 때문에 선택되었다. 그리고 본 논문에서 제안한 기법은 TCC와 LCOM 뿐만 아니라 LCC[5], Co[11], Coh[6] 등과 같은 다른 응집도 척도에도 유사한 방식으로 적용될 수 있다. 또한, 본 논문에서는 쓰기 의존성을 고려한 응집도 측정을 지원하는 도구를 개발하였으며, 이 도구를 C++ 클래스 라이브러리에 적용한 사례를 통하여 쓰기 의존성의 고려 여부가 응집도 측정에 미치는 영향을 소개한다.

본 논문의 구성은 다음과 같다. 2절에서는 TCC와 LCOM에 대한 기존의 정의를 소개한다. 그리고, 3절에서는 쓰기 의존성의 영향을 고려하도록 개선된 TCC와 LCOM을 정의한다. 4절에서는 사례 연구를 통하여 쓰기 의존성의 영향을 소개하고 마지막으로, 5절에서는 결론과 향후 연구 방향을 기술한다.

2. TCC 및 LCOM의 원 정의

2.1 기본 정의

이 절에서는 *TCC* 및 *LCOM*을 통일된 관점에서 논하기 위해 기본이 되는 몇 가지 용어를 정의한다.

정의 1. 임의의 클래스 C에 대하여 $V(C)$는 C에 구현되어 있는 인스턴스 변수의 집합을 나타내고 $M(C)$는 C에 구현되어 있는 메소드의 집합을 나타낸다.

정의 2. 클래스 C의 인스턴스 변수들, 즉 $V(C)$의 원소는 자연수에 하나씩 대응시킬 수 있으며 따라서 $v_1, v_2, \ldots v_n$(여기서 $n = |V(C)|$)과 같이 셀 수 있다. 이 때, 서로 다른 변수 쌍의 집합 $V_p(C)$를 다음과 같이 정의한다.

$$V_p(C) = \{(v_i, v_j) \mid v_i \in V(C), v_j \in V(C), i < j\}$$

마찬가지로 클래스 C의 메소드들도 $m_1, m_2, \ldots m_l$(여기서 $l = |M(C)|$)과 같이 셀 수 있으며, 서로 다른 메소드 쌍의 집합 $M_p(C)$도 다음과 같이 정의할 수 있다.

$$M_p(C) = \{(m_i, m_j) \mid m_i \in M(C), m_j \in M(C), i < j\}$$

정의 3. 클래스 C의 메소드 $m \in M(C)$에 대하여 $V_U(m)$은 m에서 직접 참조되는 C의 인스턴스 변수들의 집합을 나타내며 $M_U(m)$은 m에서 직접 호출되는 C의 메소드들의 집합을 나타낸다.

정의 4. 클래스 C의 메소드 m에 대하여 $V_R(m)$은 m에 의해 읽혀지는 인스턴스 변수들의 집합을 나타내고 $V_W(m)$은 m에 의해 쓰여지는 인스턴스 변수들의 집합을 나타낸다. V_U, V_R, V_W에 대해 다음과 같은 식이 만족된다.

$$V_R(m) \cup V_W(m) = V_U(m)$$

2.2 TCC

원래 *TCC* 척도는 모든 가능한 메소드 쌍 수에 대한 관련있는 메소드 쌍 수로 정의된다.두 메소드에서 공통적으로 사용하는 인스턴스 변수가 있을 때 두 메소드는 관련있는 메소드로 정의된다. 원 *TCC* 정의는 다음과 같다.

정의 5.

$$TCC(C) = 2\frac{|\{(m_i, m_j) \mid (m_i, m_j) \in M_p(C), V_U(m_i) \cap V_U(m_j) \neq \varnothing\}|}{|M(C)| \times (|M(C)| - 1)}$$

$M_p(C)$ 자체가 클래스 C의 서로 다른 메소드의 쌍으로 정의되었기 때문에 $(m_i \neq m_j)$와 같은 조건이 필요없다는 사실에 주의하자. *TCC*에 대한 실제적인 감각을 얻기 위해 간단한 예를 살펴보자. 클래스 C에 대해 $V(C)$와 $M(C)$ 간의 의존성을 나타내는 그래프를 참조 그래프(reference graph)라고 한다[7]. 참조 그래프의 노드(node)는 $V(C) \cup M(C)$로 정의되며 에지(edge)는 읽기/쓰기 연산에 따른 의존성을 나타낸다. 메소드 $m \in M(C)$이 인스턴스 변수 $v \in V(C)$를 읽거나 쓰면 에지 (m, v)가 참조 그래프에 추가된다. 그림 1은 참조 그래프의 예를 나타낸다.

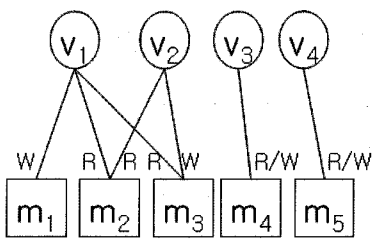


그림 1 클래스 C에 대한 참조 그래프 예

그림 1에 따르면, 클래스 C는 4개의 인스턴스 변수와 5개의 메소드를 포함하고 있다. 각 에지에는 의존성 종류에 따라 레이블이 붙어 있다. 예를 들어, m_3는 v_1에 읽기 연산을 수행하고 v_2에 쓰기 연산을 수행한다. 다른 예로, m_4는 v_3에 읽기 연산과 쓰기 연산을 모두 수행한다. 이 때, m_3는 v_1에 대해 읽기 의존성을 갖고 있다고 하고 v_2에 대해 쓰기 의존성을 갖고 있다고 한다. m_4는 v_3에 대해 읽기/쓰기 의존성을 갖고 있다.

TCC 척도를 산출하기 위해서는 참조 그래프의 의존

성을 기반으로 하여 관련있는 메소드 쌍을 산출해야 한다. m과 m'이 공통적으로 의존하고 있는 변수가 있다면 m과 m'은 관련있다고 한다. 그림 1의 경우, 관련있는 메소드 쌍은 모두 세 개($|\{(m_1, m_2), (m_1, m_3), (m_2, m_3)\}| = 3$)이고 총 메소드 쌍 개수는 열 개($|M_p(C)| = (5\times4)/2 = 10$)이다. 따라서 $TCC(C) = 3/10$이다.

2.3 LCOM

원래 *LCOM*[9] 정의에서는 메소드 쌍을 두 개의 집합, P, Q로 분류한다. P는 관련없는 메소드 쌍의 집합이며 Q는 관련있는 메소드 쌍의 집합이다. 관련없는 메소드 쌍의 수 $|P|$가 관련 있는 메소드 쌍의 수 $|Q|$보다 클 때 *LCOM*은 양의 값 $|P|-|Q|$로 정의되며, 그렇지 않으면 0으로 정의된다. 클래스 내 모든 메소드가 어떤 변수도 참조하지 않을 경우에 *LCOM*에서는 $P = \varnothing$이라고 간주한다. 원 *LCOM* 정의는 다음과 같다.

정의 6.

$$P = \begin{cases} \varnothing, \ \forall m \in M(C). V_U(m) = \varnothing \text{인 경우} \\ \{(m_i, m_j) \mid (m_i, m_j) \in M_p(C), V_U(m_i) \cup V_U(m_j) = \varnothing\}, \text{ 그 외의 경우} \end{cases}$$

$$Q = \{(m_i, m_j) \mid (m_i, m_j) \in M_p(C), V_U(m_i) \cup V_U(m_j) \neq \varnothing\}$$

라고 하면,

$$LCOM(C) = \begin{cases} |P|-|Q| & |P| > |Q| \text{인 경우} \\ 0 & \text{그 외의 경우} \end{cases} \quad (1)$$

*LCOM*은 반대 척도임에 주의하자. 다시 말해서, 클래스 C의 응집도가 C'보다 높다면 $LCOM(C) < LCOM(C')$이 된다. 또한 *LCOM*은 정규화된 척도(normalized measure)가 아니라는 사실에도 주의하자. 구체적인 예로, 그림 1과 같은 상황을 생각해 보자. 관련 없는 메소드 쌍의 수는 7개($|P| = |\{(m_1, m_4), (m_2, m_4), (m_3, m_4), (m_1, m_5), (m_2, m_5), (m_3, m_5), (m_4, m_5)\}| = 7$)이고 관련있는 메소드 쌍의 수는 3개($|Q| = |\{(m_1, m_2), (m_1, m_3), (m_2, m_3)\}| = 3$)이다. 따라서 $LCOM(C) = 7-3 = 4$이다.

3. 개선된 TCC와 LCOM 정의

이 절에서는 쓰기 의존성을 반영하여 개선된 *TCC*와 *LCOM* 척도를 제안한다. 우선, 읽기 연산에 비하여 쓰기 연산에 보다 높은 가중치가 부여되어야 하는 지에 대한 설명을 한다. 그리고 쓰기 연산에 가중치를 부여하는 방법과 이를 반영하여 *TCC*와 *LCOM* 척도를 정의하는 것을 설명한다. 그리고 쓰기 연산에 대한 가중치가 읽기 연산과 동일한 경우에 개선된 *TCC*와 *LCOM* 척도가 원래의 척도와 일치됨을 보인다.

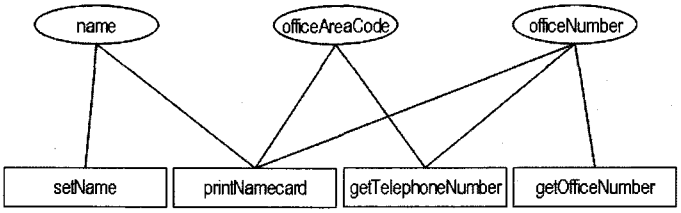
3.1 읽기 연산과 쓰기 연산의 가중치

이 절에서는 쓰기 연산에 대한 가중치 부여의 필요성을 간단한 예를 통하여 보이고자 한다. 예를 들어 그림 2는 Person 클래스와 Book 클래스에 대한 코드를 보여준다. 클래스의 멤버 간의 쓰기/읽기 연산만이 응집도 측면에서 중요하므로 이 코드에서는 각 메소드가 인스턴스 변수를 접근하는 방식(쓰기 또는 읽기)만을 표현하였다.

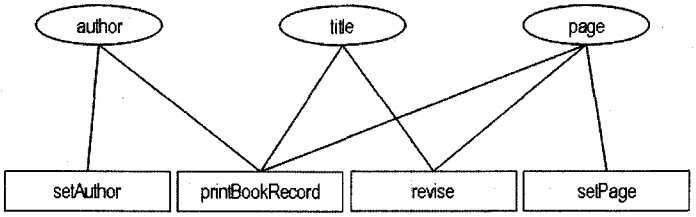
또한 그림 3은 각 클래스의 멤버 간의 상호작용을 나타내는 참조 그래프이다. 그림에서 볼 수 있듯이 두 클래스는 동일한 형태의 상호작용을 하고 있다. 따라서 기존의

```
class Person {
private:
    string name, officeAreaCode, officeNumber ;
public:
    void setName(string newName) { /* write name */ }
    void printNameCard() { /* read name, officeAreaCode, officeNumber */ }
    void getTelephoneNumber() { /* read officeAreaCode, officeNumber */ }
    void getOfficeNumber(string newNumber) { /* read officeNumber */ }
};
class Book {
private:
    string author, title ; int page ;
public:
    void setAuthor(string newAuthor) { /* write author */ }
    void printBookRecord() { /* read author, title, page */ }
    void revise(string newTitle, int newPage) { /* write title, page */ }
    void setPage(int newPage) { /* write page */ }
};
```

그림 2 Person 클래스와 Book 클래스



(a) Person 클래스



(b) Book 클래스

그림 3 Person과 Book 클래스의 참조 그래프

응집도 척도에 따르면 두 개의 클래스는 동일한 응집도 값을 갖는다. 예를 들어 *TCC* 척도에 따르면 가능한 메소드 쌍의 수는 6(= (4×3)/2)이고, 관련이 있는 메소드 쌍의 수는 4이므로 $TCC(Person) = TCC(Book) = 2/3$이 된다.

본 논문에서는 클래스 멤버 간의 상호작용의 유형을 반영하여 클래스의 특성을 구별하고자 한다. 두 클래스가 동일한 형태의 상호작용이 있다 하더라도 멤버 간의 상호작용 유형에 따라서 클래스의 특성이 크게 달라질 수 있다. 예를 들어 *Person* 클래스의 경우에는 *Book* 클래스에 비하여 비교적 쉽게 재구성이 될 수 있다.

그림 4는 Person 클래스의 일부 멤버가 *Telephone-Number* 클래스로 분리된 리팩토링(Extract Class 리팩토링)을 보여준다[20]. 그러나 동일한 리팩토링 기법을 동일한 형태의 상호작용을 가진 *Book* 클래스에 적용하는 것은 용이하지 않다. 이는 기존의 응집도로는 두 클래스의 특성(예컨대, 리팩토링의 적용의 적절성 여부)을 구별하기에는 부족함을 의미한다.

본 논문에서는 상호작용의 유형을 고려하여 동일한 형태의 상호작용을 하는 클래스 간의 특성을 구별해보고자 한다. 구체적으로 말하면 메소드가 인스턴스 변수를 참조한다는 수준이 아니라 참조의 형태(읽기/쓰기)를 응집도의 정의에 반영하고자 한다. 예를 들어 그림 5는 *Person* 클래스와 *Book* 클래스의 참조 그래프에 클래스의 멤버간의 참조 유형(W:쓰기, R:읽기)을 반영한 것을 보여준다. 그림에서 볼 수 있듯이 *Book* 클래스는 *Person* 클래스에 비하여 보다 많은 쓰기 유형을 가지고 있다.

본 논문에서는 쓰기 연산이 읽기 연산보다 더 영향력이 높을 수 있음을 응집도에 반영하고자 한다. 한 메소드가 인스턴스 변수에 쓰기 연산을 수행하는 경우 이 메소드는 해당 인스턴스 변수를 읽는 모든 메소드에 간접적으로 영향을 미칠 수가 있다. 따라서 쓰기 연산은 그 결과가 다른 많은 메소드에 의해 참조되는 경우 읽기 연산에 비하여 더 영향력이 높을 수가 있다. 본 논문

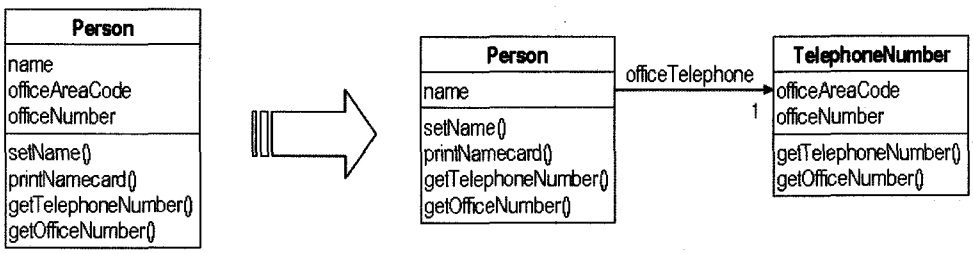
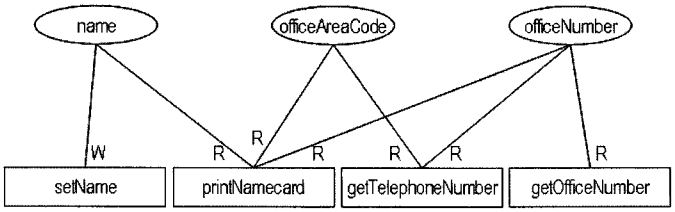
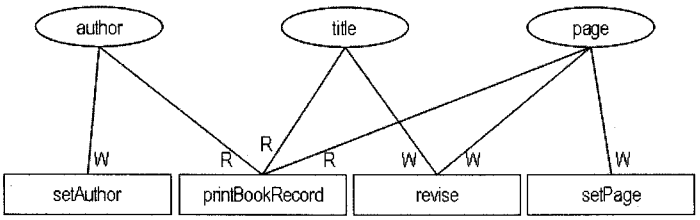


그림 4 Person 클래스의 리팩토링



(a) Person 클래스



(b) Book 클래스

그림 5 참조 유형을 구별한, Person과 Book 클래스의 참조 그래프

에서는 쓰기 유형의 상호작용에 가중치를 둠으로써 기존의 응집도 척도를 개선하고자 한다.

3.2 기본 정의

쓰기 연산을 고려하기 위해 기본적으로 참조 그래프의 모든 에지에 응집도 가중치(coherency weight)를 부여한다. 그러고 나서, 이 가중치로부터 메소드 쌍의 가중치를 산출한다. *TCC*와 *LCOM* 척도는 메소드 쌍의 가중치를 바탕으로 새롭게 정의된다. 이 절에서는 응집도 가중치를 정의하기 위한 몇 가지 기본 정의와 응집도 가중치 정의를 기술한다.

정의 7. 클래스 C의 인스턴스 변수 v, 즉 $v \in V(C)$인 v에 대해서, $M_R(v)$는 v를 직접 읽는, 클래스 C의 메소드 집합을 나타내고 $M_W(v)$는 v를 직접 쓰는, 클래스 C의 메소드 집합을 나타낸다.

정의 8. 메소드 $m \in M(C)$과 인스턴스 변수 $v \in V(C)$에 대해서, m과 v 사이의 응집도 가중치 $W_{MV}(m,v)$는 다음과 같이 정의된다.

$$W_{MV}(m,v)=\begin{cases}|M_R(v)-\{m\}|+1 & m \in M_W(v)\text{인 경우}\\ 1 & m \in M_R(v)\text{이고 } m \notin M_W(v)\text{인 경우}\\ 0 & \text{그 외의 경우}(\ m \notin M_U(v)\ )\end{cases} \quad (2)$$

위와 같이 응집도 가중치를 정의한 이유를 간단히 설명하겠다. 기본적으로 메소드 m과 인스턴스 변수 v 사이의 응집도 가중치는 m이 v를 읽거나 쓸 경우에 부여된다. 따라서 $m \notin M_U(v)$일 때 응집도 가중치 $W_{MV}(m,v)$는 0으로 정의된다. m이 v를 읽기만 할 경우에 m과 v 사이의 응집도 가중치는 1로 정의된다. m이 v에 쓰기 연산을 수행할 경우에 m과 v 사이의 응집도 가중치는 (v를 읽는 메소드의 수) + 1로 정의된다. 여기서 1을 더해 준 이유는 쓰기 연산 자체를 1로 간주하기 위해서이다. 만약 m이 v를 읽기도 하고 쓰기도 할 경우($m \in M_W(v) \cup M_R(v)$)에는 m을 중복하여 고려하게 되는데, 이를 방지하기 위해 $M_R(v)$에서 m을 빼주도록 정의하였다.

정의 9. 클래스 C의 서로 다른 메소드 쌍 $(m_i, m_j) \in M_p(C)$에 대하여 m_i와 m_j의 응집도 절대 가중치(absolute coherency weight) $W_{MMabs}(m_i, m_j)$는 다음과 같이 정의한다.

$$W_{MMabs}(m_i,m_j)=\begin{cases}0 & V_U(m_i)\cap V_U(m_j)=\varnothing\text{인 경우}\\ max\ W_{com} & \text{그 외의 경우}\end{cases}$$

여기서 $W_{com}=\{W_{MV}(m,v) \mid m \in m_i, m_j, v \in V_U(m_i)\cap V_U(m_j)\}$

두 메소드 m_i와 m_j 사이의 응집도 절대 가중치는 메소드와 인스턴스 변수 사이의 응집도 가중치를 이용하여 정의한다. m_i와 m_j가 공통적으로 사용하는 변수가 없는 경우, m_i와 m_j의 응집도 절대 가중치는 0으로 정의한다. 그렇지 않으면, 공통 변수 v ($v \in V_U(m_i) \cap V_U(m_j)$)와 m_i 또는 m_j 사이의 응집도 가중치의 최대값으로 정의한다. 이렇게 정의된 가중치 W_{MMabs}는 클래스 내의 메소드 수에 따라 값이 크게 달라질 수 있으므로 절대 가중치라고 명명하였다.

정의 10. 클래스 C에 대하여, 클래스 C 내의 메소드 간 최대 응집도 절대 가중치(maximum absolute coherency weight between methods) WMMmax(C)

는 다음과 같이 정의한다.

$W_{MMmax}(C) = max(1 \cup \{W_{MMabs}(m_i, m_j) \mid (m_i, m_j) \in M_p(C)\})$

$W_{MMmax}(C)$는 C의 메소드 사이의 응집도 절대 가중치의 최대값이다. 이 값은 메소드 간 응집도 절대 가중치를 정규화 할 때 사용할 예정이므로, 최소값이 1이 되도록 정의하였다.

정의 11. 클래스 C의 서로 다른 메소드 쌍 $(m_i, m_j) \in M_p(C)$에 대하여 m_i와 m_j의 응집도 가중치(coherency weight) $W_{MM}(m_i, m_j)$는 다음과 같이 정의한다.

$$W_{MM}(m_i, m_j) = \frac{W_{MMabs}(m_i, m_j)}{W_{MMmax}(C)}$$

메소드 사이의 응집도 가중치 W_{MM}은, 최대 절대 가중치 $W_{MMmax}(C)$를 이용하여 정규화된 응집도 절대 가중치 W_{MMabs}로 정의한다. 위에서 언급한 바와 같이, W_{MMmax}는 정규화 과정에서 제수(divider)로 사용되므로 0이 되어서는 안 된다. 따라서 W_{MMmax}는, 절대 응집도의 최대값이 0인 경우에도 1이 되도록 정의하였다.

3.3 TCCw 및 LCOMw

개선된 TCC인 TCC_w는, 서로 다른 메소드 쌍의 수에 대한 메소드 간 응집도 가중치의 합으로 정의된다.

정의 12.

$$TCC_w(C) = 2\frac{\sum_{(m_i, m_j) \in M_p(C)} W_{MM}(m_i, m_j)}{|M(C)| \times (|M(C)| - 1)}$$

예로 그림 1의 참조 그래프를 다시 한 번 생각해 보자. 참조 그래프의 에지에 대한 응집도 가중치는 그림 6(a)와 같이 계산할 수 있다. m_1은 v_1에 대해 쓰기 의존성을 갖고 있고 v_1은 총 세 개의 메소드에 연결되어 있으므로 $W_{MV}(m_1, v_1) = 3$이다. 마찬가지로, $W_{MV}(m_3, v_2) = 2$이다. 메소드 사이의 절대 가중치 W_{MMabs}는 그림 6(b)와 같이 계산할 수 있다. 이 경우에는 $W_{MMmax} = 3$이므로 W_{MM}은 그림 6(c)와 같이 계산할 수 있다. 그림 6에서 응집도 가중치가 0인 경우에는 가중치를 나타내지 않았다.

그림 6(b)에 나타낸 메소드 간 응집도 가중치 W_{MM}에 따라 응집도 가중치 합을 계산하면, 응집도 가중치 합은 8/3이다($\Sigma_{(m_i, m_j) \in M_p(C)} W_{MM}(m_i, m_j) = 8/3$). 총 메소드 쌍 수는 10이므로, 이 경우에 $TCC_w(C) = 8/30$이다.

구체적인 사례로서 앞서 소개된 *Person* 클래스와 *Book* 클래스의 경우 $TCC_w(Person) = 5/12 \approx 0.42$, $TCC_w(Book) = 2/3 \approx 0.67$이 된다. 이는 본 논문에서 제시한 개선된 TCC_w를 통하여 *Person* 클래스와 *Book* 클래스의 특성을 구별될 수 있음을 보여 준다. 앞서 언급한 것처럼 두 클래스는 동일한 상호작용 형태를 가지고 있지만 멤버 간의 상호작용의 유형을 고려할 때 *Book* 클래스가 *Person* 클래스 보다 높은 응집도를 가지는 것으로 판단할 수 있다.

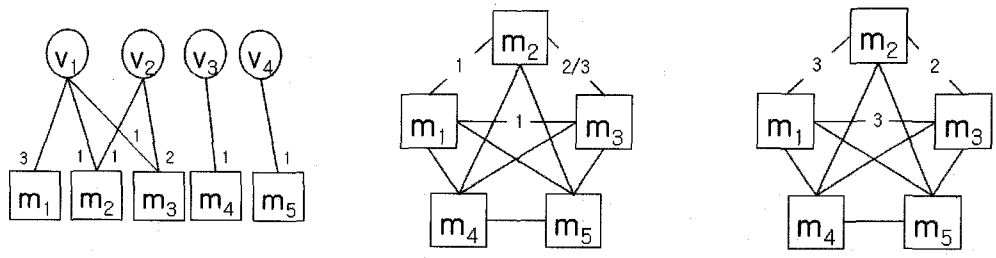
응집도 가중치를 이용하여 *LCOM*도 새롭게 정의할 수 있다. 원래 *LCOM*에서는 메소드 쌍 수만 고려하였지만 향상된 *LCOM*인 $LCOM_w$에서는 메소드 사이의 응집도 가중치를 고려한다.

정의 13.

$$LCOM_w(C) = \begin{cases} |M_p(C)| - 2\Sigma_{(m_i, m_j) \in M_p(C)} W_{MM}(m_i, m_j) & |P| > |Q| \text{인 경우} \\ 0 & \text{그 외의 경우} \end{cases}$$

$LCOM_w$ 정의는 $LCOM$ 정의와 같은 구조로 정의되어 있다. $|P| \le |Q|$인 경우 $LCOM_w$는 0으로 정의한다. 그 외의 경우($|P| > |Q|$)에 $LCOM_w$는 모든 메소드 쌍 수에서 관련있는 메소드 쌍의 응집도 가중치 합의 두 배를 뺀 것으로 정의된다.

그림 1의 경우, 제2.3절에서 계산한 바와 같이 $|P| = 7$, $|Q| = 3$이다. 따라서 위 $LCOM_w$ 정의의 첫 번째 경우를 적용해야 한다. 총 메소드 쌍 수는 10($|M_p(C)| = 10$)이고 응집도 가중치의 합은 8/3($\Sigma_{(m_i, m_j) \in M_p(C)} W_{MM}(m_i, m_j) = 8/3$)이므로 $LCOM_w$는



(a) W_{MV} (b) W_{MMabs} (c) W_{MM}

그림 6 클래스 C에 대한 응집도 가중치

$LCOM_w(C) = 10-2 \times 8/3 = 14/3$이다.

3.4 축약 가능성

이 절에서는 원래의 *TCC* 및 *LCOM* 척도와 새로 정의된 TCC_w 및 $LCOM_w$ 척도 사이의 관련성을 기술한다. 보다 구체적으로 말해서, *TCC* 및 *LCOM*이 TCC_w 및 $LCOM_w$에서 쓰기 연산에 대한 가중치를 1로 둘 경우 *TCC* 및 *LCOM*으로 축약됨을 보인다.

이를 위해, 먼저 응집도 척도 사이의 축약 관계를 정의한다. 응집도 척도는 읽기 연산에 대한 가중치 *RW*와 쓰기 연산에 대한 가중치 *WW*에 따라 정의된다. 이들 가중치는 클래스 *C*의 참조 그래프 에지에 부여되었었다. 응집도 척도 사이의 축약 관계는 다음과 같이 정의된다.

정의 14. 응집도 가중치 매개변수 $W_i \in \{RW, WW\}$를 특정 값 n으로 변경함으로써 응집도 척도 M으로부터 응집도 척도 M'이 유도된다면, $W_i = n$으로 설정함으로써 M'은 M으로부터 축약 가능하다고 하고 이 관계를 다음과 같이 쓴다.

$$M \xrightarrow{W_i = n} M'$$

이제 우리는 $TCC_w \xrightarrow{WW=1} TCC$이고

$LCOM_w \xrightarrow{WW=1} LCOM$임을 보이고자 한다.

사실 1. WW = 1일 때, 응집도 가중치 사이에는 다음과 같은 관계가 성립한다.

$$W_{MV}(m, v) = \begin{cases} 1 & m \in M_U(v)\text{인 경우} \\ 0 & m \notin M_U(v)\text{인 경우} \end{cases} \quad (3)$$

$$W_{MMabs}(m_i, m_j) = \begin{cases} 0 & V_U(m_i) \cap V_U(m_j) = \varnothing\text{인 경우} \\ 1 & \text{그 외의 경우} \end{cases} \quad (4)$$

$$W_{MMmax}(C) = 1 \quad (5)$$

$$W_{MM}(m_i, m_j) = W_{MMabs}(m_i, m_j) \quad (6)$$

식 (2)에서 $m \in M_W(v)$인 경우가 바로 쓰기 연산에 대한 가중치를 정의하고 있는 부분이다. 따라서 $WW=1$일 때 이 경우의 가중치는 1로 정의될 수 있으며 두 번째 경우($m \in M_R(v)$이고 $m \notin M_W(v)$인 경우)와 함께 정의될 수 있다. 결과적으로 식 2는 식 3와 같이 간략화된다. 다른 모든 응집도 가중치(W_{MMabs}, W_{MMmax}, W_{MM})는 W_{MV}를 기반으로 하여 정의되므로 식 (4), (5), (6)은 간단히 유도된다.

정리 1. $WW=1$로 설정함으로써 *TCC*는 TCC_w로부터 유도 가능하다. 즉, 다음 관계가 성립한다.

$$TCC_w \xrightarrow{WW=1} TCC$$

증명. $WW=1$일 때, 메소드 사이의 응집도 가중치의 합은 관련있는 메소드 쌍의 수가 된다.

$$|\{(m_i, m_j) \mid (m_i, m_j) \in M_p(C), V_U(m_i) \cap V_U(m_j) \neq \varnothing\}| = \sum_{(m_i, m_j) \in M_p(C)} W_{MM}(m_i, m_j)$$

따라서 $WW=1$일 때, $TCC(C) = TCC_w(C)$

$$\therefore \quad TCC_w \xrightarrow{WW=1} TCC \qquad \square$$

정리 2. $WW=1$로 설정함으로써 *LCOM*은 $LCOM_w$로부터 유도 가능하다. 즉, 다음 관계가 성립한다.

$$LCOM_w \xrightarrow{WW=1} LCOM$$

증명. *LCOM*과 $LCOM_w$는 경우에 따라 다르게 정의되므로 이 증명에서도 각 경우에 대해 성립함을 보인다.

(i) $|P| \leq |Q|$ 인 경우

$$LCOM_w(C) = 0 = LCOM(C)$$

(ii) $|P| > |Q|$ 인 경우, $WW=1$이라면 다음 식이 성립한다.

$$|P| + |Q| = |M_p(C)| \quad (7)$$

$$|Q| = \Sigma_{(m_i, m_j) \in M_p(C)} W_{MM}(m_i, m_j) \quad (8)$$

따라서, $LCOM_w(C) = |M_p(C)| - 2|Q|$ 식 (8)에 의해
$= |P| + |Q| - 2|Q|$ 식 (7)에 의해
$= LCOM(C)$ 식 (1)에 의해

(*i*)과 (*ii*)에 의해서, $LCOM_w \xrightarrow{WW=1} LCOM$ □

4. 사례 연구

이 절에서는 본 논문에서 제안한 기법을 클래스 라이브러리에 적용한 사례 연구를 소개한다. 우리는 본 논문의 기법을 지원하는 클래스 응집도 측정 도구를 개발하였으며 이 도구를 InterViews[21] 라이브러리에 적용하였다. InterViews는 Stanford 대학에서 개발된 C++ 클래스 라이브러리로서 X 윈도우 상에서의 GUI를 제공하기 위한 많은 클래스로 구성되어 있다.

개발된 도구는 *TCC*와 LCOM의 원래 정의와 본 논문에서 쓰기 의존성을 고려하도록 개선된 정의를 모두 계산할 수 있다. 도구는 입력 받은 C++ 소스 프로그램을 분석하여 클래스를 구성하는 멤버 및 멤버 간의 상호작용, 즉 메소드와 인스턴스 변수 간의 읽기/쓰기 의존성을 구한다. 그리고, C++ 프로그램으로부터 이와 같은 정보를 추출할 때는 GEN++[22] 도구를 이용하였다. 우리는 InterViews 시스템의 107 개의 클래스를 대상으로 실험을 하였으며, 그림 7은 InterViews 시스템에서 쓰기 의존성이 차지하는 비중을 보여 준다.

그림에서 첫 번째 막대(57.7 %)는 클래스의 멤버 간에 존재하는 모든 상호작용에서 쓰기 의존성 즉 쓰기 상호작용의 비율을 보여 준다. 즉, InterViews 시스템에서는 클래스 멤버 간의 상호작용의 반 이상은 읽기가 아니라 쓰기라는 것을 알 수 있다. 그림의 두 번째 막대(45.3 %)는 한 개 이상의 쓰기 의존성을 가지고 있는 클래스의 비율을 보여준다. 즉, InterViews의 전체 클래스 중에서 거의 반 정도의 클래스가 한 개 이상의 쓰기 의존성을 포함하고 있음을 보여 준다. 비록 InterViews라는 하나의 시스템에 대한 분석 결과 이지만, 이 데이타를 통해 쓰기 의존성이 실제 객체지향 프로그램에서 빈번히 나타나는 것을 알 수 있으며 쓰기 의존성에 대한 고려 여부가 응집도 측정에 많은 영향을 미칠 수 있음을 알 수 있다.

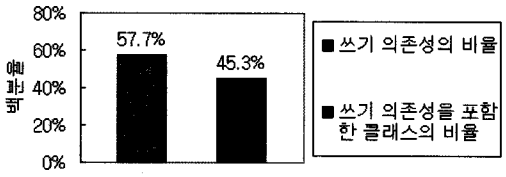
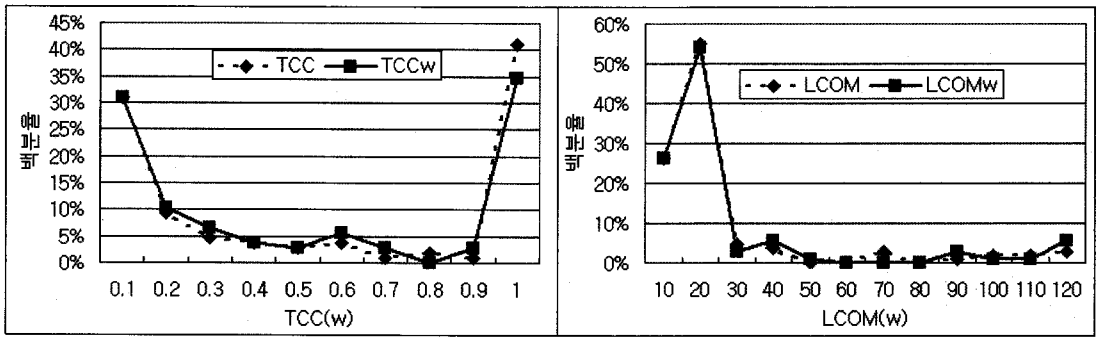


그림 7 InterViews 시스템에서의 쓰기 의존성

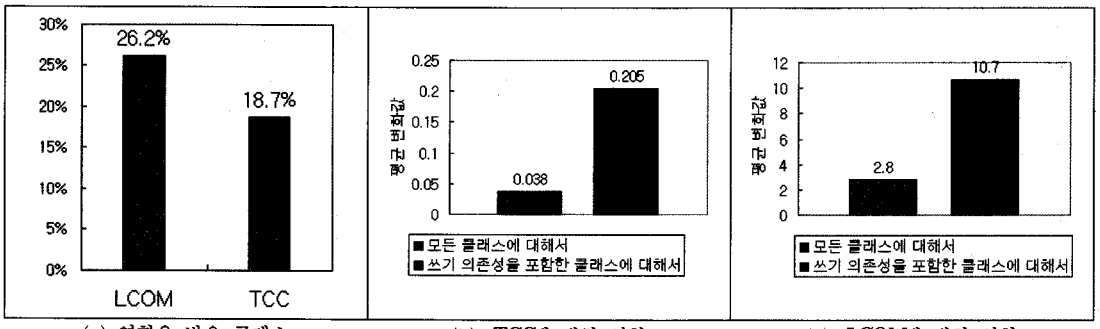
그림 8은 InterViews 시스템에 대해서 측정된 TCC, TCC_w와 $LCOM$, $LCOM_w$ 값의 분포를 나타낸다. 그림 8(a)는 TCC와 TCC_w의 분포로서, 이 그림을 통해 TCC_w가 TCC보다 전반적으로 작다는 것을 알 수 있다. 그리고 그림 8(b)는 $LCOM$과 $LCOM_w$의 분포를 나타내는데, 이 그림에서는 $LCOM$과 $LCOM_w$ 간에 일관된 어떤 패턴을 발견할 수는 없었다. 그러나 어느 경우든, 클래스 멤버 간의 쓰기 의존성을 고려하였을 때 TCC과 $LCOM$의 값이 변화함을 확인할 수 있다.

그림 9는 쓰기 의존성의 고려 여부가 응집도 측정에 미치는 영향을 보여 준다. 그림 9(a)는 $LCOM$과 TCC에 대해서 쓰기 의존성을 고려하였을 때 응집도 값이 변경된 클래스의 비율을 보여 준다. 즉, $LCOM$의 경우에는 26.2 %의 클래스가 응집도 측정 값이 변경되었으며 TCC의 경우에는 18.7 %의 클래스가 응집도 값이 달라졌다. 그림 9(b)는 TCC 값의 평균 변화량을 보여준다. 첫 번째 막대(0.038)는 InterViews의 모든 클래스를 대상으로 할 때 측정된 응집도 값의 평균 변화량을 나타내며, 두 번째 막대(0.205)는 한 개 이상의 쓰기 의존성을 가진 클래스를 대상으로 할 때의 응집도 값의



(a) TCC와 TCC_w (b) $LCOM$과 $LCOM_w$

그림 8 TCC_w와 $LCOM_w$의 분포



(a) 영향을 받은 클래스 (b) TCC에 대한 영향 (c) $LCOM$에 대한 영향

그림 9 쓰기 의존성에 의한 응집도 값의 변화

평균 변화량을 나타낸다. *TCC*의 값은 0에서 1까지의 값으로 정규화된 값을 가지므로, 두 번째 막대가 보여주는 0.205 값으로부터 쓰기 응집도의 고려 여부에 따라서 *TCC*의 값의 변화가 큼을 알 수 있다. 그림 9(c)는 *LCOM*에 대해서 응집도 변화의 평균 값을 보여 준다. 첫 번째 막대(2.8)와 두 번째 막대(10.7)는 각각 InterViews의 모든 클래스 또는 한 개 이상의 쓰기 의존성을 가진 클래스를 대상으로 하였을 때의 *LCOM* 값과 $LCOM_w$ 값과의 차이의 평균 값을 나타낸다. 그림 9(c)에서도 쓰기 의존성을 고려하였을 때 *LCOM* 값이 상당히 변화함을 알 수 있다.

5. 결 론

본 논문에서는 객체지향 프로그램에서 쓰기 의존성이 클래스 응집도에 미치는 영향을 소개하고 이를 반영하여 기존의 응집도 척도를 개선하는 방법을 소개하였다. 그리고, *TCC*와 *LCOM*을 대상으로 쓰기 의존성을 고려하도록 개선된 응집도 척도를 소개하였다. 또한, 사례 연구를 통하여 쓰기 의존성을 고려하였을 때 응집도 측정 값의 변화를 살펴봄으로써 쓰기 의존성의 고려 여부가 응집도 측정에 미치는 영향을 살펴보았다.

향후 연구로는, 본 논문에서 제안한 방법을 LCC[5], Coh[6], CBMC[7] 등과 같은 다른 응집도 척도에도 적용해 보는 것과 또 다른 시스템을 대상으로 사례 연구를 수행함으로써 쓰기 의존성의 고려 여부에 대한 중요성을 더 살펴보는 것을 들 수 있다. 또한, 응집도 척도뿐만 아니라, 결합도 척도에도 쓰기 의존성을 고려해 볼 수 있겠다. 그리고 쓰기 의존성을 고려하여 개선된 응집도 척도가 원래의 응집도 척도에 비하여 클래스의 품질 요소(오류 발생 정도, 유지보수도 등)와 더 밀접한 관계를 가지는 지에 대한 실험적인 연구를 수행해 볼 수 있다.

참 고 문 헌

[1] W. Stevens, et. al, "Structured Design," *IBM Systems Journal*, Vol. 12, No. 2, 1974.

[2] N. N. Card, et. al, "Criteria for Software Modularization," *Proc. of 8th Int. Conf. on Software Engineering*, pp. 372-377, 1985.

[3] N. N. Card, et. al, "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 OOPSLA*, pp. 84-91, 1986.

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

[6] L. C. Briand, et. al, "A Unified Framework for Cohesion Measurement in Object-Oriented Systems," *Empirical Software Engineering Journal*, Vol. 1, No. 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," *Proc. 6th ACM Conf. on OOPSLA*, 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, No. 6, 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," *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, No. 2, pp. 111-122, 1993.

[13] H. -S. Chae, Y. -R. Kwon and D. -H. Bae, "Improving Cohesion Metrics by Considering Dependent Instance Variables," *IEEE Trans. on Software Engineering*, Vol. 30, No. 11, pp. 826-832, 2004.

[14] Y. Zhou, et. al, "DRC: A Dependence Relationships Based Cohesion Measure for Classes," *Proc. of the 10th IEEE APSEC*, pp. 215-223.

[15] Y. Zhou, et. al, "A Comparative Study of Graph Theory-based Class Cohesion Measure," *ACM SIGSOFT SE Notes*, Vol. 29, No. 2, March 2004.

[16] L. C. Briand, et. al, "Investigation of Quality Factors in Object-Oriented Designs: An Industrial Case Study," *Proc. of Int. Conf. Software Engineering*, pp. 345-354, 1999.

[17] L. C. Briand, et. al, "Exploring the Relationship between Design Measures and Software Quality in Object-Oriented Systems," *Journal of Systems and Software*, Vol 51, No. 3, pp. 245-273, 2000.

[18] L. C. Briand and J. K. Wust, "Modeling Development Effort in Object-Oriented Systems Using Design Properties," *IEEE Trans. on Software Engineering*, Vol. 27, No. 11, pp. 963-986, 2001.

[19] M. Bruntink and A. Deursen, "Predicting Class Testability using Object-Oriented Metrics," *Proc. of 4th Int. Workshop on Source Code Analysis and Manipulation*, 2004.

[20] M. Folwer, *Rectoring: Improving the Design of Existing Code*, Addison Wesley, 1999.

[21] M. Linton, et. al, "InterViews: A C++ Graphical

Interface Toolkit," *Technical Report CSL-TR-88-358*, Stanford University, 1988.
[22] P. Devanbu, "GENOA a customizable, language- and front-end independent code analyzer," *Proc. Conf. on Software Engineering*, pp. 307-317, 1992.

우 균

1991년 한국과학기술원 전산학 학사. 1993년 한국과학기술원 전산학 석사. 2000년 한국과학기술원 전산학 박사. 2000년~2002년 동아대학교 컴퓨터공학과 전임강사. 2002년~2004년 동아대학교 컴퓨터공학과 조교수. 2004년~현재 부산대학교 컴퓨터공학과 조교수. 관심분야는 프로그래밍언어 및 컴파일러, 함수형 언어, 그리드컴퓨팅, 소프트웨어 메트릭

채 홍 석

정보과학회논문지 : 소프트웨어 및 응용 제 32 권 제 7 호 참조