런타임을 고려한 소프트웨어 컴포넌트 메트릭스

차석기[O] 임정은[1] 백두권[1]
[O]고려대학교 컴퓨터·정보통신대학원 소프트웨어공학과, [1]고려대학교 정보통신대학 컴퓨터학과
{stonecha, jelim, dkbaik}@korea.ac.kr

A Software Component Matrix in Run-Time Environment

Seok-Ki Cha[O] Jung-Eun Lim[1] Doo-Kwon Baik[1]
[O]Dept. of Software Engineering, Korea University
[1]Dept. of Computer Science & Engineering, Korea University

요 약

소프트웨어 컴포넌트는 빠르게 변화하는 컴퓨팅 환경과 시대의 추세에 신속히 대응할 수 있도록 개발의 가능성을 이루게 하는 소프트웨어의 단위이다. 이는 소프트웨어 컴포넌트 간의 의존성을 줄이고 응집력을 높이는 것을 핵심으로 한다. 이러한 소프트웨어 컴포넌트를 정량적으로 평가할 수 있는 중요한 지표는 바로 결합도와 응집도이다. 본 논문에서는 런타임 상황을 고려하여 클래스 추상화 정도에 따른 클래스의 응집력을, 소프트웨어 컴포넌트 응집력으로 확장한다. 또한 컴포넌트 인터페이스에 의한 내부 결합도와 컴포넌트 간의 의존성에 따른 외부 결합도 측정법을 제안한다. 본 논문에서는 제안 메트릭스를 사례에 적용하여 그 효율성을 평가한다

1. 서 론

소프트웨어 모델은 현실을 단순화한다[1]. 여기서 단순화한다는 것은 시스템의 엔터티나 사물의 본질을 모델링하는 것을 의미한다. 다시 말해서 이는 분할과 정복(Divide and Conquer)의 원리를 이용하여 주어진 문제를 적당한 크기로 분할하고, 분할된 문제를 기능적으로 분류하여 해결해 나가는 원리이다. 이러한 원리를 바탕으로 소프트웨어 컴포넌트는 제기되는 문제들을 해결해 나간다. 요컨대 이는 기능적 분류를 정의 할 수 있는 단위이자, 독립적인 행위를 할 수 있는 소프트웨어의 단위이다. 한편 이는 빠르게 변화하는 컴퓨팅 환경에 신속히 대응하기 위해 재사용성·확장성·유지보수의 용이성을 가져야 하는데, 이 지점에서 응집도와 결합도가 요구된다. 소프트웨어 컴포넌트의 인터페이스는, 기능적으로 분류된 각 클래스들이 현실세계에서의 비즈니스 업무 환경 및 흐름에 맞추어 객체의 생성과 삭제 그리고 메소드를 호출할 수 있도록 깊숙이 관여한다. 여기서 클래스들 간의 관계, 즉 객체 기술에 기반을 둔 소프트웨어 컴포넌트는 클래스들 간의 결합뿐만 아니라 컴포넌트 간의 결합을 낮추고 응집력을 높이는 관점이 중요해진다. 응집도(Cohesion)와 결합도(Coupling)는 클래스 및 컴포넌트 구성 요소들 간의 관계를 측정할 수 있는 소프트웨어 컴포넌트의 중요 구성 요소이다. 1979년 Edward & Larry[9]는 소프트웨어 시스템의 유지보수성과 적응성을 측정하기 위해 응집도(Cohesion)와 결합도(Coupling)의 각각 중요한 원리인 High Cohesion, Loose Coupling을 제안하였는데 이것은 현재의 소프트웨어 시스템에서 일반적인 가치를 지닌다[9]. 응집도란 소포트웨어 구성 요소들이 서로 얼마나 기능적으로 밀접하게 집적되어 있는가를 나타내는 척도이며 결합도는 소프트웨어 구성 요소들이 서로 얼마나 밀접하게 결합되어 있는가를 측정하는 척도이다. 그러므로 이러한 요소들은 소프트웨어 컴포넌트를 설계하고 식별하는 과정에서 생기는 결점과 오류를 극복할 수 있도록 도와준다. 본 논문에서는 런타임 상황을 고려하여 클래스 추상화 정도에 따른 소프트웨어 컴포넌트의 응집력 측정 방법을 제안한다. 또한 컴포넌트 인터페이스에 의한 내부 결합도와 컴포넌트 간의 의존성에 따른 외부 결합도 측정법을 동시에 제안한다. 이를 위해 본 논문의 구성은 2장에서는 기존 측정법의 문제점을 기술하고, 3장에서는 클래스 추상화에 따른 컴포넌트 응집도 결합도 측정 방법을 제안한다. 4장에서는 사례 연구를 통하여 제안된 측정 방법을 평가하고 5장에서는 결론 및 향후 연구과제를 제시한다

2. 관련연구 및 연구 배경

2.1 관련연구

Chidamber and Kemerer[2]가 제안한 LCOM, CBO, RFC, DIT, NOC는 클래스 기반의 객체지향 매트릭스이다.

객체지향 매트릭스

LCOM(Lack of Cohesion in Methods)은

LCOM = $|P| - |Q|$, if $|P| > |Q| = 0$ otherwise이다.

단, $P = \{(I_i, I_j) \mid I_i \cap I_j = \emptyset\}$ $Q = \{(I_i, I_j) \mid I_i \cap I_j \neq \emptyset\}$

여기서 I_j: 메소드 j가 사용하는 변수의 집합

클래스의 메소드는 공통 인스턴스 변수를 사용하는

쌍의 수와 사용하지 않는 쌍의 차로서 정의하는데, 그 값이 0보다 작으면 LCOM 값은 0이라 한다. 이는 메소드들이 클래스내의 인스턴스 변수들을 많이 사용한다면 클래스의 응집도가 높다는 것을 뜻한다. 그러나 이 측정법은 상속에 대한 부분을 고려하지 않았으며 응집도의 측정값을 비율적으로 계산하지 않았다. 따라서 이를 응집도를 측정할 수 있는 값으로 판단하기에는 불충분하다[2]. CBO(Coupling BetWeen Object Class)는, 하나의 클래스가 다른 클래스의 메소드나 속성을 사용하여 결합된 클래스의 수를 측정한다. CBO는 상속이 없는 클래스에 해당하는데, 특히 CBO값이 클 경우 클래스들 간에 맺어지는 관계가 복잡하게 증가하는 것을 예측해 볼 수 있다[2]. RFC(Response For a Class)는 클래스 안에 선언된 메소드 안에서 다른 클래스의 메소드를 호출하여 클래스의 반응 정도를 측정한다[2]. DIT(Depth of Inheritance Tree)는 상속의 깊이를 측정하는 척도로써 상속받은 클래스가 상위 클래스에서 얼마나 많은 영향을 받았는지를 측정한다[2].

NOC(Number of Children)는 직접 상속된 서브 클래스의 수가 어느 정도 인지를 측정하고, 하위 클래스가 상위 클래스로부터 메소드를 상속받는 척도를 제시해준다[2]. Li and Henry는 DIT, NOC를 이용한 상속 결합도와 추상 자료형을 기반으로 결합도를 정의하였다[3]. MPC(Message Passing Coupling)는 하나의 클래스가 다른 클래스의 메소드의 호출 수를 가지고 결합도를 측정하는 방법이다[4]. Choi[5]는 클래스 기반의 종속적인 관계를 이용한 정적관계와 유즈 케이스에 기반한 동적 결합도를 이용하여 컴포넌트의 결합도·응집도를 제안하였다. ko[6]는 공통으로 사용하는 클래스에 대한 정보를 활용하여 오퍼레이션의 사용도 행렬을 구하고 이를 통해 컴포넌트의 응집도와 결합도 측정을 제안하였다.

기존의 소프트웨어 컴포넌트의 응집도와 결합도를 측정하는 방법은 클래스 기반의 정적인 관계-합성·상속·연관·종속-에서 비롯되는 가중치와, 유즈 케이스를 중심으로 하는 객체의 생성과 삭제 그리고 메소드 호출 유형에 따른 가중치를 고려하였다. 클래스 기반의 정적인 관계 중 상속과 합성의 측정 방법에 대해 기존의 논의는 다음과 같다. 상속은 상위 클래스가 변경을 할 때 하위 클래스에 영향을 끼쳐 이 때 생겨나는 강한 의존성으로 인해 같은 컴포넌트에 포함 되어져야 한다고 정의 되었다[5][7][8]. 논의된 사항들은 클래스간의 높은 결합도를 갖게 되어 컴포넌트의 크기가 커질 수 있는 가능성이 있다. 이로 인해 하나의 클래스를 변경 하려고 하면 컴포넌트의 복잡성으로 유지보수의 어려움이 존재 한다. 합성관계는 강력한 결합력을 갖지만 추상화클래스 포인터를 유지한다면 종속성을 줄일 수 있다. 종속성 문제는 클래스 상속이 컴파일 시점에 정적으로 결정되기 때문이다. 그러므로 클래스 추상화를 통해서 종속성의 문제는 해결이 가능하다. 즉 선언과 구현을 완전히 분리 하여 종속성을 제거 하는 것이다.

기존의 유즈 케이스를 기반으로 하는 동적 결합력 측정 방법은 객체가 생성될 때 런타임 상황이나 객체지향의 폴리모피즘을 고려하지 않았다. 이에 따라 프로그램 내부의 조건문에 맞추어 객체를 생성하는 소스 코드를 작성하게 되는 문제가 발생될 수 있고, 아울러 인터페이스의 숫자가 증가하거나 인터페이스 시그너처의 일관성을 떨어뜨리는 문제점들이 발생될 수 있다.

2.2 컴포넌트 매트릭스의 필요성

컴포넌트 기반의 소프트웨어 개발 방법은 다음과 같은 기대 효과가 있다. 첫째 검증된 컴포넌트를 가지고 개발하기 때문에 품질 개선의 효과가 있을 뿐만 아니라 생산성 향상의 효과를 가져다 준다. 둘째 컴포넌트 신뢰성을 확보하고 있기 때문에 어플리케이션 개발의 신뢰도를 향상시킬 수 있다. 세째 표준화된 EJB나 COM+ Corba 기술을 사용하기 때문에 다양한 컴퓨팅 환경에서의 활용을 가능케 한다[10]. 이러한 컴포넌트의 특징으로 인해 컴포넌트를 설계하거나 식별할 때, 이를 정량적으로 평가할 수 있는 지표가 매우 중요한 사항으로 떠오른다. 그러나 기존에는 컴포넌트 가지고 있는 고유의 특성 보다는 객체 기술 기반의 매트릭스를 활용하는 데에 치중했다. 그러나 컴포넌트는 객체 기술 기반 위에서 만들어지는 방법론이다. 따라서 객체 지향 특성과 컴포넌트의 특성이 결합된 컴포넌트 매트릭스에 대한 연구가 필요하다.

2.3 런타임 상황을 고려한 컴포넌트 매트릭스

인터페이스 상속은 OOP(ObjectOrinted Programming)에서 중요한 개념 중 하나이다. 객체는 인터페이스를 통해서 자기 자신의 행동양식을 드러내고, 다른 기능적 분류로 나누어진 외부 시스템에 접근하기 위해 노출된 인터페이스를 사용한다. OOP 특징인 인터페이스 상속을 통해서 동일한 인터페이스를 갖지만, 객체의 성격에 따라 완전히 다른 형태의 구현이 가능하다. 그리고 인터페이스 상속은 프로그램 코드를 작성할 때 일관성을 유지할 수 있도록 도움을 준다. 이러한 메커니즘을 객체지향에서는 폴리모피즘(Polymorphism)라고 하고, 더불어 폴리모피즘은 런타임이 이루어지는 시간에 객체를 결정하는 기법인 동적 바인딩(Dynamic Binding)을 가능하게 한다. 결국 폴리모피즘은 클래스의 인터페이스를 추상화하여 응집력은 높고 결합력이 낮은 컴포넌트를 개발할 수 있는 토대를 마련하는 기틀을 제공할 수 있다. 그러므로 런타임 상황을 고려하여 소프트웨어 컴포넌트의 응집도와 결합도를 제안하였다

3. 런타임을 고려한 컴포넌트 매트릭스

3.1 클래스 추상화에 따른 컴포넌트 응집도

컴포넌트 응집도는 해당 컴포넌트에 포함되는 클래스 그룹들의 추상화 정도에 의존한다. 따라서 추상화 정도

에 의존하는 공통성을 추출하고, 이를 클래스 인터페이스로 선택할 것인지를 판단하는 것은 매우 중요하다.

[정의1]클래스의 인터페이스는 최상위 클래스에서 순수 가상 함수로 정의된 것을 말하고, 컴포넌트 인터페이스는 비니지스 흐름에 따라 객체들의 메소드 호출과 생성에 관여하는 메소드를 말한다. 그 외에 구현에 관련한 함수는 메소드라고 정의한다.

[정의2] 추상화 클래스 인터페이스 (Abstract Class)
상위 클래스인 추상화 클래스는 기능적 분류의 공통적 특성을 반영한 인터페이스 $I_1, I_2, I_3, I_4, \cdots I_n$ 를 구성하고 있다. 이를 바탕으로 추상화 클래스를 아래와 같이 정의할 수 있는데, 여기서 인터페이스는 구현을 하지 않은 순수 가상 함수이고 최상위 클래스는 객체를 선언할 수 없는 순수 가상 클래스이다.

$$AC_n = \{I_1, I_2, I_3, I_4, I_5, \ldots I_{n-1} I_n\}$$

[정의3] 구현 클래스 (Implement Class)
구현 클래스는 최상위 순수 가상 클래스에서 인터페이스를 상속받아 구성된 클래스로써 아래와 같이 정의할 수 있다.

$$IC_n = \{AC_n\} \cup \{DC_n \mid DC_n \text{ is personalized}\}$$
DC_n = 구현 클래스 만의 메소드 집합

[정의4] 클래스 추상화에 따른 클래스 응집도
(GCCH: a Group Class of Cohesion by Abstract Interface)

$$GCCH = \frac{\sum_{k=1}^{n}(AC_k / IC_k)}{N-1}$$
N = 총 클래스 갯수

상속 받은 클래스는 상위 클래스의 인터페이스 수와 자기 자신만의 특성을 가진 메소드 수의 합을 갖는다. 이것을 IC_n라고 부른다. 한편 AC_n는 상위클래스의 인터페이스 수를 의미한다. 하위클래스는 어느 정도로 공통 메소드를 재정의 할 수 있는지에 대한 비율을 의미한다. 각각의 구현 클래스의 비율을 합하여 구현 클래스의 개수로 나누게 되면 추상화 그룹의 평균 응집력을 구할 수 있다. 이 과정 속에서 GCCH의 값이 커지는 경우가 발생한다. GCCH의 값이 크다는 것은 추상성이 높을 뿐만 아니라 기능적 분류가 잘되어 있다는 것을 의미한다. 이는 런타임이 이루어지는 시간에 같은 타입의 변환을 이루게 하고 같은 이름의 메소드를 호출하도록 한다. 한편 클래스 개수에서 -1을 한 것은 추상클래스가 객체 생성을 할 수 없기 때문이다.
[정의5] 클래스 추상화에 따른 컴포넌트 평균 응집도
(ACCH: a Average Component Cohesion by GCCH)
컴포넌트는 비지니스의 유사성에 따라 추상 클래스 그룹이 모여 이루어진 집합으로 구성된다. ACCH는 각각의 추상화 클래스 그룹의 응집력의 합을 전체 클래스 추상 클래스 그룹의 개수로 나누어 구해진 컴포넌트의 평균 응집력이다. (아래 수식 참조) 이처럼 소프트웨어 컴포넌트는 추상화가 높은 클래스로 구성되기 때문에 응집도가 높다. 또한 컴포넌트의 인터페이스 호출에 따른 클래스의 호출 관계도 단순하여 유지 보수에 용이하다.

$$ACCH = \frac{\sum_{k=1}^{n} GCCH_k}{N}$$
N =추상 클래스 그룹의 갯수

3.2 추상화의 따른 컴포넌트 결합도
[정의6] 컴포넌트 인터페이스(Component Interface)
컴포넌트 인터페이스는 컴포넌트가 가지는 행동양식을 나타내는데 이는 아래와 같이 정의할 수 있다.

$$CI_n = \{CI_1, CI_2, CI_3, CI_4, CI_5, \ldots CI_{n-1} CI_n\}$$

[정의7]컴포넌트 인터페이스의 내부 결합도
(ICC: a Inner Coupling of Component)
컴포넌트 인터페이스의 내부는 메소드의 호출 형태에 따라 Edward & Larry[9]가 정의한 결합 형태에 가중치를 적용하며 구성된다. 또한 컴포넌트는 더 큰 모듈의 결합 관계이기 때문에 클래스 간에 호출 가중치의 실험적 근거에 의하여 2배를 적용한다.

식별	결합도 유형	클래스간 가중치	컴포넌트 가중치
I	Message coupling	0.1	0.2
II	Data coupling	0.2	0.4
III	Stamp coupling	0.3	0.6
IV	Control coupling	0.4	0.8
V	External coupling	0.5	1.0
VI	Common coupling	0.6	1.2
VII	Content coupling	0.7	1.4

[표1] 결합도 유형에 따른 가중치

Ⅰ)의 경우 공개된 인터페이스나 두 모듈 간에 아무런 영향 없이 서로 호출하므로 가장 낮은 결합도를 갖는다. II)의 경우 모듈 사이의 인터페이스는 Primitive 데이터 파라미터로 전달이 되며 변경 내용은 존재하지 않는다. III)의 경우 모듈 간에 같은 자료 구조를 공유하는 형태이다. IV)의 경우 파라미터를 통해 다른 모듈의 흐름을 제어하는 형태로, 정보를 받은 모듈은 실행에 영향을 받는다. V)의 경우 다른 모듈에서 정의해 놓은 자료 형태

를 간접적인 형태 그대로 사용해야 하므로 모듈 간의 의존도가 높다. Ⅵ)의 경우 모듈은 전역에 선언한 데이터를 참조한다. 따라서 데이타를 참조하는 모듈에 직접적으로 영향을 준다. Ⅷ)의 경우 외부에 선언이 되지 않고 해당 모듈의 안에 선언된 데이터를 직접 참조하게 됨으로써 강한 결합력을 얻어 다른 모듈에 영향력을 크게 끼친다. Ⅰ은 가장 결합력이 낮고 Ⅷ은 가장 모듈 간의 결합력이 높다[9].

컴포넌트 인터페이스의 내부는 비지니스 흐름에 따라 구현된 클래스의 객체 생성과 소멸에 관여할 뿐만 아니라, 메소드의 호출, 객체의 상태 변경, 데이터 참조, 객체의 포인터 등의 전달을 런타임이 이루어지는 동안 가능케 한다. 따라서 컴포넌트 인터페이스의 내부 결합력은 메소드 호출, 즉 결합 유형에 의존한다. 따라서 아래와 같이 제안된 수식을 통하여 컴포넌트 인터페이스의 내부 결합도를 측정한다.

$$ICC = \sum_{i=1}^{n} CI_i \left(\sum_{j=1}^{k} \text{Class Object} \rightarrow \text{Method}_j * \text{Weight} \right)$$

Weight는 [표1]에 의한 가중치

각 구해진 인터페이스는 [표1] 결합도 가중치의 값을 곱하여 합을 구하게 되면 컴포넌트의 내부 결합도를 구할 수 있다

[정의8] 컴포넌트 외부 결합도
(OCC: a out Coupling between Component)

하나의 컴포넌트는 컴포넌트의 기능을 사용하기 위해서 컴포넌트의 객체를 생성한 후 인터페이스를 호출한다. 컴포넌트 간의 상호 작용이 많아지게 되면 컴포넌트의 응집도가 낮아지고 반면에 컴포넌트의 결합도는 높아진다. 요컨대 컴포넌트의 변경이 이루어지면, 컴포넌트 간에 연쇄적으로 수정하는 현상이 발생하여 컴포넌트를 유지하고 보수하는 어려움을 초래한다. 따라서 외부 결합도 측정은 컴포넌트의 간의 상호작용을 수치화함으로써 소프트웨어 컴포넌트의 이해성, 식별성, 수정 가능성을 측정 할 수 있다.

$$OCC = \sum_{i=1}^{n} CO_i(\text{Componet Object} \rightarrow \text{Interface} * \text{Weight})$$

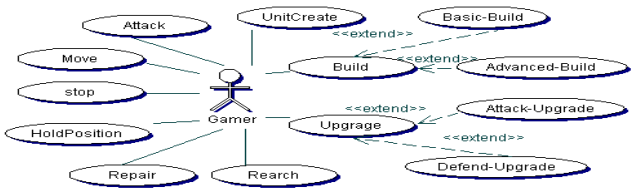
Weight는 [표1]에 의한 가중치

4. 사례 연구 및 비교 평가

아래의 그림은 전략 시뮬레이션 게임인 스타크래프트의 유즈 케이스이다. 이 게임은 마우스와 키보드를 사용하여 유닛을 통제하고 생산할 뿐만 아니라 적과의 교전을 하는 전략 시뮬레이션 게임이다. 이 게임을 대상으로 하여 제안한 방법으로 런타임 상황을 고려한 컴포넌트의 응집도와 결합도를 측정하고 그 유효성을 비교 평가하기로 한다.

4.1 사례모델

[그림1]는 대략적인 게임의 구조를 설계한 클래스 다이어그램이다. 이 클래스 다이어그램에서 유닛이 가져야 하는 공통적인 특성들을, IUnit 최상위클래스에 interface로 정의하였다.



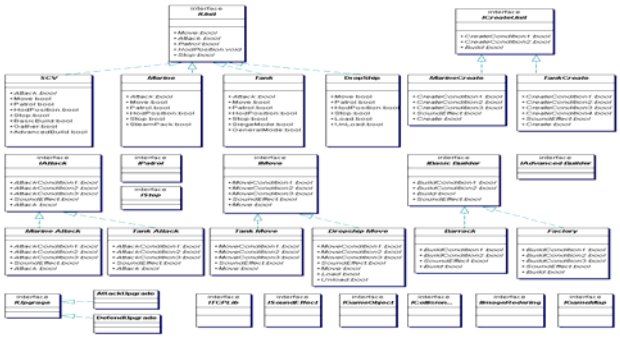
[그림1]사례모델의 유즈케이스

[표2]은 [그림2]을 기반으로 최상위 클래스의 인터페이스 ID를 정의하였다.

IUnit	A	IUpgrade	H
IAttack	B	IGameObject	I
IMove	C	ICollision	J
IPatorl	D	IImageRender	K
ICreateUnit	E	IGameMap	L
IBasicBuilde	F	ISoundEffect	M
IAdvancedBuilder	G	ITcpIpLiB	N

[표2]추상화 클래스와 ID 목록

아래의 [표3]는 대략적인 게임 구조에 맞추어 행위, 유닛 생산, 구조물 생산, 유닛 업그레이드, 기타 게임 관련 컴포넌트를 만든다는 가정 하에 컴포넌트에 포함될 후보 클래스를 정의한 사례이다.



[그림2] 클래스 다이어그램

컴포넌트	ID	구현 Class
Unit(1)	A	SCV,Marine,Tank, DropShip
	B	ScvAttack, MarineAttack,

행위		TankAttack,
	C	DropShipMove, MarineMove,TankMove
	D	ScvPatorl, MarinePatorl, TankMove
Unit(2) 생산	E	MarineCreate,TankCreate SCVCreate,
구조물(3) 생산	F	Barrack,Academy
	G	Factory,StarPort
GameObject(5)	I	IGameObject
Collision(6)	J	ICollision
사운드(7)	K	ISoundEffect
맵관련(8)	L	IGameMap

[표3]컴포넌트와 클래스 후보 ID

4.2 클래스 추상화에 따른 클래스 응집도

아래의 표는 각 컴포넌트 후보 클래스의 응집도를 [정의4]를 바탕으로 도출된 값이다. 응집도는 인터페이스의 추상화 정도에 따라 응집력이 높고 낮음이 결정된다. 따라서 이 응집도가 높을수록 다른 클래스와 결합할 확률이 적어진다.

A	응집도	B	응집도
SCV	0.625	SCVAttack	0.6
Marine	0.833	MarineAttack	0.8
ITank	0.714	TankAttack	0.666
DropShip	0.714	FireBatAttack	0.8

[표4] 클래스 추상화에 따른 클래스 응집도

구분	추상화 그룹	평균응집도
①	IUnit	0.721
②	IAttack	0.716
③	IMove	0.677
④	IPatrol	0.714
⑤	ICreateUnit	0.55

[표5] 후보 추상클래스의 그룹 응집도

4.3 추상화에 따른 컴포넌트 응집도 측정

추상화 그룹에 의한 평균 응집도를 구한 후 전체 컴포넌트의 평균을 얻으면 컴포넌트의 응집도를 구할 수 있다.

컴포넌트	식	컴포넌트 응집도
Unit 행위	①+②+③+④/4	0.707
Unit 생산	⑤	0.55

[표6] 컴포넌트 응집도

4.4 컴포넌트의 결합도

4.4.1 컴포넌트 내부 결합도 측정

Gamer가 Unit을 마우스로 선택하여 공격 명령을 내렸을 경우 게임 UI는 해당 객체의 타입을 런타임이 이루어지는 시간에 결정해 준다

컴포넌트	인터페이스
Unit 행위컴포넌트	IUnit IAttack IMove IPatrol

가령 유저가 Marine 유닛을 선택하여 공격 명령을 내렸을 경우, attack 인터페이스는 Marine_Attack 객체를 생성하고 메소드를 호출한다. 그리고 사운드 관련 컴포넌트의 컴포넌트 오브젝트를 생성하고 SoundEffect() 메소드에서 Marine의 사운드 효과 함수를 호출한다. 또한 AttackCondition 함수 내부에서는 IGameMap을 참조한다.

ID	I	II	III	IV	V	VI	VII
A	–	–	–	–	–	–	–
B	1	3	1	–		–	–

[표7]attack 내부 클래스를 참조한 매트릭스

attack 인터페이스의 내부 결합도는 함수 호출 유형에 따라 가중치를 곱해서 더하여 아래와 같이 산출할 수 있다.

1 * 1.01 + 3 * 1.02 + 1 * 1.03 = 5.10

구분	Unit 행위 컴포넌트	내부 결합도
①	attack	5.10
②	move	4.96
③	patrol	4.75

[표8] Unit 행위 인터페이스의 내부 결합도

따라서 내부 결합도 5.10+4.96+4.75 = 14.81 이다.

4.4.2 컴포넌트의 외부 결합도 측정

아래 [표9]는 attack, move, patrol 인터페이스가 다른 컴포넌트의 인터페이스를 사용 하였을 경우 참조하는 컴포넌트 인터페이스 호출 횟수이다.

ID	I	II	III	IV	V	VI	VII
(5)	3	–	–	–	–	–	–
(7)	–	3	–	–	–	–	–
(8)	–	3	–	–	–	–	–

[표9] 컴포넌트를 참조한 메트릭스

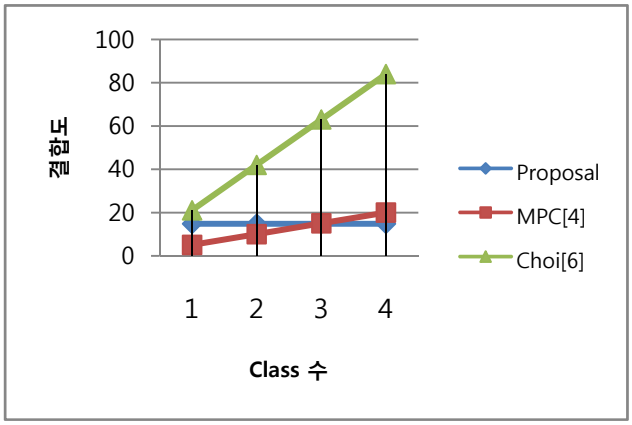
아래의 표는 두개의 컴포넌트 결합도와 응집도를 산출하였다.

컴포넌트	외부결합도	내부결합도	응집도
Unit 행위	9.3	14.81	0.707
Unit 생산	1.4	4.04	0.55

[표10] 컴포넌트 응집도 결합도

4.5 비교평가

비교연구에서는 Choi[6], MPC[4]와 본 논문에서 제안한 컴포넌트 메트릭스의 결합도 측정을 비교한다. 4.4절의 사례연구의 유닛행위 컴포넌트의 Attack 인터페이스를 대상으로 한다. 그림[3]은 클래스 수가 1개 일때 Choi[6]와 제안된 메트릭스는 가중치에 의한 결합도값이 비슷하다. 하지만 같은 기능을 하는 클래스 수가 증가함에 따라 메소드 호출 수와 결합도는 선형적으로 증가 한다. 즉 폴리모피즘을 고려 하지 않았기 때문에 런타임 시 컴포넌트 내부에서 객체의 타입이 동적으로 변경되지 않고 완전히 새로운 타입의 메소드가 호출 되기 때문이다. 따라서 제안된 응집도, 내부 외부 결합도는 추상화 따른 공통성 추출 및 런타임의 컴포넌트간의 가중치를 고려하여 실제 업무에서 적용 하였을 경우 기존의 메트릭스 보다 품질, 식별성, 수정예측성, 컴포넌트간의 결합도, 응집도 측정이 용이함을 알 수 있었다.



[그림 3]결합도 비교

[표 11]은 기존의 메트릭스와 제안된 메트릭스의 비교 결과를 제시한다[5][6][7][8].

비교요소	기존 메트릭스	제안 메트릭스
가중치	고려됨	고려됨
가중치의 근거	클래스 C,R,U,D	Edward Larry[9]
컴포넌트간 가중치	고려되지 않음	고려됨
클래스간 가중치	고려됨	고려되지 않음
메소드 호출 유형	고려됨	고려됨
메소드 호출 수	고려됨	고려됨
추상화 상속	고려되지 않음	고려됨
구현 상속	고려됨	고려되지 않음
컴포넌트의 특성	부분적 고려	고려됨
컴포넌트내부결합도	구분하지 않음	고려됨
컴포넌트외부결합도	구분하지 않음	고려됨
컴포넌트 응집도	고려됨	고려됨
응집도 측정방법	클래스 간의 유사도, 긴밀도	객체지향의 추상화 비율
런타임 상황	고려되지 않음	고려됨
폴리모피즘	고려되지 않음	고려됨

[표11] 기존 컴포넌트 메트릭스와 제안 메트릭스 비교

5. 결론 및 향후 연구 과제

본 논문에서는 인터페이스 상속을 통한 추상화된 클래스 그룹의 응집력을 측정하였다. 그리고 어플리케이션에서 런타임이 이루어지고 컴포넌트의 인터페이스를 사용할 때, 동적인 객체 타입 변환과 메소드 유형에 따른 가중치를 적용하여 컴포넌트 내·외부 결합도를 측정하였다. 이처럼 제안된 내용을 검증하기 위하여 본 논문에서는 사례 연구를 통하여 그 결과를 제시하였다. 우선 런타임을 고려한 클래스 설계는 컴포넌트의 품질 향상과 유연성을 향상시킬 수 있었다. 다음으로 컴포넌트를 정량적으로 평가함으로써 분석 설계 단계 시에 비용과 시간을 절감은 실무에 적용 함으로써 가능함을 알 수 있었다. 향후 객체 지향의 특성과 컴포넌트 특성을 좀 더 세분화 하는 것이 필요하며 이에 대한 연구가 필요하다.

참고문헌

[1] James Rumbaugh, Ivar Jacobson, Grady Booch "The Unified Modeling Language User Guide" Addison-Wesley, 1998

[2] S.R Chidamber and C.F Kemerer, "A Metric Suite For Object-Orinted Design", IEEE Transaction On Software Engineering, Vol.17, No.6, pp636-638, 1994

[3] Li, Henry S, "Object-Orinted Metrics That Predict Maintainability, " Journal of System and Software, Vol. 23, pp.111-122, 1993

[4] Henderson-Sellers,Brain,"Object-Orinted Metrics" Prentice-Hall, 1996

[5] 최미숙, 이종성, 송행숙, "컴포넌트 설계를 위한 결합도 매트릭" 한국정보처리학회 논문지 D, 제 12-D권, 제4호 pp.609-616, 2005

[6] 고병선, 박재년, "컴포넌트 설계에 대한 응집도와 결합도 매트릭스" 한국정보처리학회논문지 D, 제10-D권 제5호, 2003

[7] J. K Lee, S.J. Jung and S.D Kim, "Component Identification Method with Coupling and Cohesion" Processing of Asia-Pacific Software Engineering Conference, pp.419-426, 2001.

[8] 최미숙, 조은숙, 박재년, 하종성, "클래스들 간의 정적 동적 관계에 의한 2단계 컴포넌트 식별방법", 한국정보과학회논문지, 컴퓨팅의 실제 제11권,1호 2005

[9] Larry Constantine과 Edward Yourdon "Structured Design: Fundamentals of a Discipline Of Computer Program and Systems Design" Prentice-Hall, 1979

[10] 한국 소프트웨어 컴포넌트 컨소시움 "컴포넌트란 무엇인가 알기쉬운 소프트웨어 컴포넌트" 2002