패키지 중심의 객체지향 코드의 컴포넌트 추출을 위한 메트릭

A Metric of Component Extraction for Package based Object Oriented Codes

이종호(Jong-Ho Lee)*, 류성열(Sung-Yul Rhew)**

초 록

컴포넌트 기반 소프트웨어 개발(CBSD: Component Based Software Development)이 재사용을 통한 소프트웨어 개발의 효율적인 방법으로 인식되고 있다. CBSD의 목적은 새로운 시스템을 개발하고자 할 때 검증된 컴포넌트, 검증된 업무 로직을 재사용 함으로써 개발 기간을 단축하고 품질을 향상하고자 하는 것에 있다.

본 논문에서는 기 존재하는 객체지향 어플리케이션을 폐기하지 않고, 이를 기반으로 재사용성이 높은 컴포넌트로 추출하기 위하여, 컴포넌트 추출을 위한 메트릭과 인터페이스 추출 방안을 제시 하였다. 클래스 간의 관계 정보를 이용하여 복잡도, 응집도와 결합도를 측정하고, 이의 결과를 이용하여 후보 컴포넌트를 추출한다.

Abstract

Component-based software development (CBSD) has been recognized effective reuse techniques for software development by many of researchers and companies. The purpose of CBSD is to produce a high quality software system quickly through using verified software component which is contained fine-grained business logics.

This paper suggests the metrics and techniques for to extract component and its interface from legacy object oriented application. For extract component, we apply metrics to measure complexity, cohesion and coupling to the legacy system.

키워드 : 소프트웨어 재사용, 컴포넌트
Software Reuse, Component

* 대신정보통신(주)
** 숭실대학교 컴퓨터학부 교수

1. 서 론

1.1 연구배경 및 목표

컴포넌트 기반 소프트웨어 개발(CBSD: Component Based Software Development)이 재사용을 통한 소프트웨어 개발의 효율적인 방법으로 인식되면서 학계 및 산업계의 기대를 모으고 있다. 이러한 컴포넌트 기반 소프트웨어 개발의 목적은 새로운 어플리케이션을 개발하고자 할 때 검증된 컴포넌트, 검증된 업무 로직을 재사용 함으로써 개발 기간을 단축하고 품질을 향상하고자 하는 것에 있다[1, 2].

하지만 이러한 컴포넌트를 개발할 때(CD: Component Development) 순공학의 순서에 따라 다양한 사용자의 요구사항을 수집하는 것으로부터 시작하여 Rational사의 RUP, Cool Software의 CBD96, ETRI의 마르미III 등의 CBD 방법론에 따라 대상 시스템에 대해 분석, 설계를 거쳐 신규 개발하게 된다. 또한 컴포넌트 기반 소프트웨어를 개발할 때에도, 기 운영중인 시스템과의 인터페이스를 연계하기가 쉽지 않아 기 시스템을 폐기하고, 전면 컴포넌트 기반으로 재 개발하는 사례가 업계에 존재하고 있다. 하지만, 실 환경에서 운영 중이며, 업무적으로 그 정확성과 성능이 검증된 어플리케이션을 보유하고 있는데 이를 재사용하지 못하고, 폐기하게 되는 것은 대규모 자원의 낭비일 수 밖에 없다[3, 4].

본 논문에서는 기 개발된 객체지향 시스템을 폐기하지 않고, 시스템 내의 코드 내 존재하는 관계정보로부터 컴포넌트 후보를 추출하여, 이를 기준으로 컴포넌트 기반 시스템으로 전이하도록 지원할 수 있도록 한다.

1.2 연구방향

기 운영중인 시스템으로부터 후보 컴포넌트 정보를 추출하거나, 이를 기준으로 컴포넌트로의 전환 및 변경시 다음과 같은 두가지 문제 상황이 발생할 수 있다.

첫번째, 추출 또는 전환하고자 하는 기 어플리케이션에 대한 현상파악에 대한 어려움이 발생할 수 있다. 기 개발된 산출물이 재사용에 적절하게 개발되어 있지 않거나, 변경에 대한 체계적 형상관리가 부족하거나, 영역 지식 및 코드 체계에 대한 개발자 의존도가 큰 경우인데, 이로 인하여, 컴포넌트로 추출될 수 있는 부분의 식별, 커스터마이징이 어려워져 개발 기간이 증가하고, 기존 코드가 더욱 복잡해지는 결과를 가져오게 된다.

두번째, 컴포넌트로 추출 또는 전환하고자 하는 대상이 되는 객체지향 시스템이 구조적인 문제나 에러를 내포하고 있을 경우 이로 인하여 추출된 컴포넌트의 품질이 저하되는 문제가 발생할 수 있다. 객체지향 시스템은 기본적으로 클래스를 중심으로 개발이 이루어진다. 즉, 문제 영역에 존재하는 중대한 대상 또는 개념을 클래스로 모델링하고, 이로부터 생성된 객체들 사이의 메시지 교환을 통해서 시스템은 구축된다. 그러나, 클래스의 초기 설계가 정확하지 못하거나, 유지보수가 진행되는 동안 원 클래스의 변경, 추가 등이 발생하게 되며 이러한 과정 중 객체지향 시스템의 품질이 저하될 수 있다[5, 6].

본 논문에서는 선술한 문제점을 극복하면서, 후보 컴포넌트의 정확성을 향상시키기 위하여 객체지향 메트릭을 적용하여, 후보 컴포넌트를 추출한다. 이를 위하여, 기존 연구에서의 컴포넌트 추출기법에 대하여 2장에서 알아보고, 이들의 문제점을 지적한다. 3장에서는 후보 컴포넌트 추출을 위한 메트릭을 제시한다. 4장에서는 Weyuker의 복잡도 성질을 이용하여 제안한 메트릭을 검증한다.

2. 관련연구

본 장에서는 기존 컴포넌트 추출기법에 대한 기존 연구에 대하여 제시하고, 기존 컴포넌트 추출 방법의 문제점을 보인다.

2.1 컴포넌트 식별방법

여러 컴포넌트 개발 방법론이 제시되고 있으나, 대부분에서 직관적이고 분석자의 경험에 의존적인 컴포넌트 식별방법만을 제공하고 있다. 본 절에는 대표적인 컴포넌트 방법론인 Catalysis와 CBD96, RUP, COMO의 컴포넌트 추출방법을 보인다.

2.1.1 Catalysis

Catalysis는 OMT의 정형화 작업에 그 뿌리를 두고 있으며, 비즈니스 모델로부터 소스코드에 까지 이르는 전 과정에서의 추적성을 보장하고, 불명확한 모델이나 문서를 최대한 명확하도록 만드는 정확성, 컴포넌트의 개발시 코드뿐 아니라 분석 및 설계의 산출물 역시 재사용 하도록 하는 재사용성, 유연한 프로세스를 그 특징으로 하고 있다[7].

모델 요소와 프로세스 패턴 등의 다양한 기법을 사용하고 있지만, 이러한 기법들의 적용을 위해서는 사용자의 많은 경험을 필요로 할 뿐더러, 컴포넌트 식별의 절차가 개념적이고, 분석자의 직관에 상당한 의존도가 있다.

2.1.2 CBD96의 기법

CBD96은 Catalysis를 기반으로 하고 있으며, 컴포넌트 모델링을 위해, 요구사항 정의, 분석, 행위명세, 아키텍쳐 모델링의 단계를 정의하고 있다[8]. CBD96에서는 컴포넌트를 모델링하기 위한 업무들은 정의되어 있으나, 업무에 대한 지침이 구체적으로 명시되어있지 않으며, 특히 도메인 모델링과 유즈케이스 모델링에서 컴포넌트를 추출하기 기법이 제시되어 있지 않다. 그리고 행위 명세 단계의 인터페이스 모델링 업무에서 인터페이스를 어떻게 정의할 것인지에 대한 지침이 빈약하다.

2.1.3 RUP의 기법

Rational사의 RUP(Rational Unified Process)에서는 다음과 같은 세가지의 컴포넌트 추출 기법을 제시하고 있다[9].

첫째는 프로세스 설계를 통한 컴포넌트 추출 기법인데, 액터가 발생시키는 이벤트를 시작점으로 하고, 순차 다이어그램을 참고하여 하나의 작업 쓰레드를 식별하게 된다. 쓰레드란 하나의 실행 흐름을 의미하는 것으로, 이의 대상으로는 한 프로그램 이나 동적 모델 또는 여러 제어 흐름의 표현 등이 속할 수 있다고 RUP에서는 정의하고 있다.

둘째는 클래스간의 결합 정도를 기반으로 추출하는 방법으로, 상호 협력 다이어그램(Collaboration Diagram), 클래스 다이어그램을 참조로 서로 밀접하게 협력하는 클래스군을 식별하게 된다. RUP에서는 컴포넌트를 사소하지 않고, 독립적이며, 시스템의 일부분에 대해 대체 가능하고, 명확한 기능을 잘 정의된 시스템 내에서 수행하며 여러 인터페이스들을 통해 서비스를 제공하는 단위라고 정의하고 있다.

셋째는 아키텍처 레벨에서 식별된 서브 시스템 기준으로 식별하는 방법으로, 시스템을 사용자에게 기능 군을 제공하는 집합으로 정의하고, 유즈케이스를 중심으로 설계한 업무 아키텍처의 하위 기능군별로 컴포넌트를 식별하도록 한다.

2.1.4 COMO의 기법

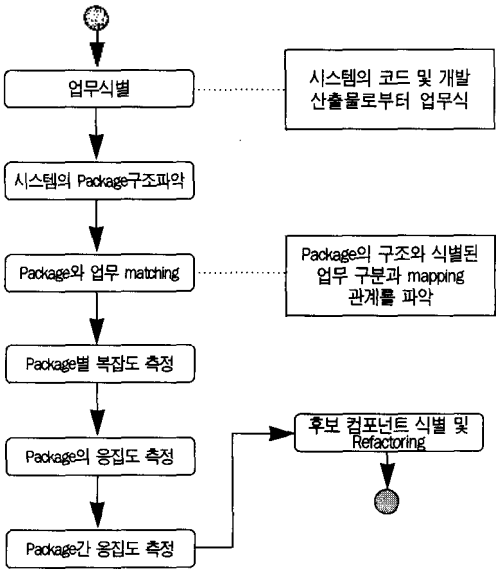
COMO에서는 컴포넌트를 추출하기 위하여 유즈케이스 간의 관계 클러스터링과 유즈케이스와 클래스간의 관계 클러스터링을 통한 두가지 기법을 제시하고 있다[10]. 특별히 유즈케이스/클래스 행렬(Matrix)를 통한 클러스터링에 초점을 맞추어 컴포넌트 식별을 설명하고 있는데, 유즈케이스와 클래스 간의 관계를 접근하는 성격에 따라 생성(Create), 수정(Write), 삭제(Delete), 참조(Read)의 4가지로 나누어 정의한다. 두 가지 이상의 관계가 가능한 경우 우선 순위가 높은 관계로 표현되며, 우선순위는 생성, 삭제, 수정, 참조의 순이다.

하지만, 공통성이 높은 클래스의 경우 많은 유즈케이스로부터 참조 되기 때문에, 컴포넌트의 크기가 지나치게 커지거나, 그 범위를 제한하는 문제를 가지고 있으며, 참조관계만 가지는 클래스, 유즈케이스의 경우 설계자의 직관에 의지할 수 밖에 없다.

3. 컴포넌트 식별을 위한 메트릭 정의

컴포넌트 간의 통신이 플랫폼의 자원과 시간의 측면에서 매우 비싼 연산이기 때문에, 일반적으로 컴포넌트의 크기는 크게 된다. 하지만, 크기가 크게 되면, 컴포넌트 및 인터페이스가 복잡해지고 변경이 될 가능성이 많기 때문에, 이를 위해서는 패키지 내에 속한 클래스 간의 관계, 패키지와 패키지 간의 관계정보에 근거하여 후보 컴포넌트를 추출, 정제하여 그 크기가 결정되어야 한다[11].

한국전자통신연구원(ETRI)의 컴포넌트 방법론인 마르미III 등 여러 방법론에서 컴포넌트의 추출은 유스케이스로부터 시작하고 있으며, 추출된 컴포넌트가 동일 도메인의 유사 업무군이라는 시각을 가지고 있다. 객체지향 개발에서 사용하는 패키지 개념은 후보 컴포넌트로 식별될 수 있다. 시스템의 설계 및 개발 시 이는 시스템의 업무 및 기능적 구분이기 때문이다. 다음 그림1에 업무를 식별 한 후 패키지를 중심으로 후보 컴포넌트를 식별하는 절차를 UML의 액티비티(Activity) 다이어그램의 표기법을 이용하여 보이고 있다.



〈그림 1〉 후보 컴포넌트 식별을 위한 절차

본 논문에서는, 다음의 순서로 후보 컴포넌트를 추출, 정제한다.

1) 시스템에서 제공하는 업무 및 기능을 식별 한다.

기존 시스템의 기능을 사용자 인터페이스, 설계/개발 문서, 산출물 등을 이용하여 파악하고, 시스템의 범위와 시스템 내부의 기능 군을 식별한다.

2) 시스템의 패키지 구조를 파악 한다.

개발 코드 및 라이브러리 정보를 이용하여, 시스템의 패키지 구조를 파악한다.

3) 패키지와 업무를 매칭한다.

시스템의 식별된 업무와 패키지를 비교하고, 어떠한 기준으로 구분 되었는지 파악한다.

4) 패키지 내의 복잡도를 측정한다.

복잡도는 사람이 소프트웨어를 대하는 심리적 복잡도이며, 이는 소프트웨어의 이해성과 직접적인 연관이 있고, 신뢰성과 유지보수성에 관하여 소프트웨어의 품질을 평가할 때 주 평가 요소로 적용된다. 복잡도는 시스템을 구성하는 모듈의 내부와 외부에 대한 관점에서 볼 때, 모듈의 내부 요소들에 의한 내부 복잡도 메트릭, 모듈과 그의 환경과의 상호작용의 의한 외부 복잡도 메트릭으로 구분할 수 있다[12].

외부 복잡도를 측정하는 대표적인 척도로는 WMC(Weighted Methods per Class), RFC(Response For a Class)가[13,14], 내부 복잡도를 위해서는 McCabe의 Cyclomatic 복잡도가 있다[15].

본 논문에서는 패키지의 내부 복잡도와 외부 복잡도를 평가할 수 있는 메트릭을 제안하며, 패키지의 복잡도가 타 패키지의 평균보다 높은 경우, 다수개의 컴포넌트로의 분리 대상으로 선정한다.

5) 패키지 내의 응집도를 측정한다.

응집도란 모듈 구성 요소들이 얼마나 연관되었는지를 측정하는 척도이므로 높은 응집도를 가진 모듈이란 하나의 기본적인 기능만을 수행하는 여러개로 나누어지지 않는 모듈을 의미한다. 그런데, 패키지는 여러 클래스 들로 구성되어 있으며, 이 클래스의 구성요소는 그 클래스 자체에서 정의된 인스턴스 변수와 메소드, 그리고 상속된 요소들이다. 따라서 패키지의 응집도를 측정한다는 것은 패키지 내에 있는 클래스 들이 얼마나 연관이 있는지 측정하는 것이 될 것이고, 이것이 클래스의 기준으로 본다

면 동일 패키지 내에 존재하는 클래스 간의 결합도를 측정하는 것과 동일하다.

결합도란 모듈이 다른 모듈과 얼마나 밀접하게 연관되어 있는지를 평가하는 것으로, 클래스 간의 결합도를 측정하기 위한 척도 사용되는 CBO(Coupling Between Object Class)는 클래스 내의 메소드가 타 클래스의 변수나 메소드를 사용하는 것으로 정의된다[13, 14]. 이 방법은 하나의 클래스의 결합도를 간단히 측정할 수 있지만, 동일 클래스의 변수나 메소드를 호출하는 경우에도 한번 사용할 때와 동일한 값을 가지는 단점이 있다. 본 논문에서는 이를 극복하기 위해, 클래스간의 관계 정보를 이용하는 정적 결합도와 클래스 간의 호출빈도를 이용하는 동적 결합도를 제안한다.

6) 패키지간 결합도를 측정한다.

선술한 대로 그 비용 때문에 컴포넌트 간의 호출은 가능한 피해야 한다. 이러한 성질을 후보 컴포넌트가 갖도록 하기 위하여, 패키지 간 결합도를 측정하여, 결합도가 높은 패키지끼리의 통합을 고려하여야 한다. 결합도는 패키지 내의 응집도를 평가할 때 사용한 클래스 간의 결합도를 그대로 적용하여 사용한다.

3.1 메트릭을 이용한 컴포넌트 식별

3.1.1 패키지의 복잡도

본 논문에서는 컴포넌트의 후보인 객체지향 어플리케이션 패키지의 복잡도를 측정하기 위하여, 이의 척도인 PCM(Package Complexity Metric)을 제안한다. PCM은 패키지를 구성하고 있는 외부적 요소인(PstC : Package Static Complexity) 클래스와 메소드의 수와 내부적 요소(PdmC : Package Dynamic Complexity)인 클래스 내의 메소드의 복잡도로 정의하며, 이를 다음과 같이 표현한다.

[정의1] PCM(P) = $PstC + PdmC$

패키지의 외부적 요소인 PstC는 패키지 내 포함된 클래스, 메소드, 애트리뷰트, 클래스 간의 관계의 수와 이들 각각의 가중치로 측정하며, 다음 정의 2와 같이 정의한다.

[정의2]

$$PstC = \sum_{i=1}^{n}(Count(C_i) \times W(C_i)) + Msgreff,C) + \sum_{i=1}^{m} CM_i + \sum_{i=1}^{o} CA_i + \sum_{i=1}^{p} CR_i$$

where :

$$CM = \sum_{i=1}^{n}(Count(M_i) \times W(M_i))$$

$$CA = \sum_{i=1}^{n}(Count(A_i) \times W(A_i))$$

$$CR = \sum_{i=1}^{n}(Count(R_i) \times W(R_i))$$

Count(C) : 패키지 내에 포함된 클래스의 수

Count(M): 패키지 내에 포함된 메소드의 수

Count(A) : 패키지 내에 포함된 애트리뷰트의 수

Count(R) : 패키지 내에 포함된 클래스 간 관계의 수

W(C) : 클래스의 가중치

W(M) : 메소드의 가중치

W(A) : 애트리뷰트의 가중치

W(R) : 클래스간 관계정보의 가중치

변수 I,j,k,l : 클래스, 메소드, 애트리뷰트, 관계정보의 수

클래스 간 관계정보의 가중치는 UML에서 정의하고 있는 4가지 관계인 의존(Dependency), 애그리게이션(Aggregation), 상속(Generalization), 컴포지션(Composition)를 사용하며, 이들의 가중치는 Dependency〈Aggregation〈Generalization〈Composition 의 순으로 부여한다[16].

애트리뷰트는 하나의 변수형태 이거나 DTO(Data Transfer Object)또는 VO(Value Object)로 정의되는 객체 형식 그리고 이들의 Collection의 형태가 존재할 수 있다.

패키지의 내부적 요소인 PdmC는 McCabe의 Cyclomatic 복잡도를 이용하며, 이는 정의3과 같이 정의되며, 패키지 내에 존재하는 클래스의 Cyclomatic 복잡도의 합으로 측정하며, Cyclomatic 복잡도는 클래스가 포함하고 있는 메소드에서 edge에서 node를 빼고 2를 더한 수로 정의된다.

[정의3] $pdmC = \sum_{i=1}^{n} CPC(C)$

where :

$CPC(C) = \sum_{i=1}^{n} (edges - nodes + 2)\ of\ class\ C$

3.1.2 패키지의 응집도

패키지의 응집도를 측정하기 위해서는 선술한대로, 동일 패키지 내에 존재하는 클래스 간의 결합도를 측정하여야 한다. 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 객체지향 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 다른 결합도와 같이 취급하여 클래스의 결합도를 측정하고자 한다.

한 클래스의 결합도를 알기 위해서는 그 클래스가 다른 클래스들과 어떤 연관을 가지고 있는지 측정해야 한다. 이를 위해 외부 참조요소를 정의 하는데, 이는 각각의 메소드가 참조하는 외부 클래스에 있는 요소들의 수를 나타낸다.

가) 클래스 관계 가중치

클래스 간의 관계에 따라서 결합성이 강한 관계에 대해 가중치를 부여할 수 있으며 가중치의 수치적인 값은 클래스 간의 접근 권한에 따라 주어 진다. UML에서는 4종류의 클래스 간 관계를 정의하고 있으며, 접근 권한에 따라 가중치를 부여하며, 이는 3.1.1절에서 정의한 Dependency〈Aggregation〈Generalization〈Composition의 규칙을 따른다.

나) 계층 및 동일 패키지에 대한 가중치

동일한 기능 군에 속한 클래스들, 동일 업무

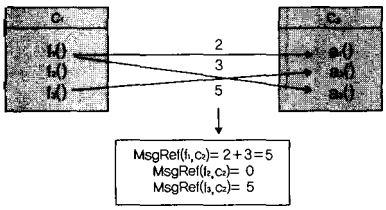
군에 해당하는 패키지 내에 존재하면 이에 대한 가중치를 두어 이를 결합도에 고려한다.

다) 결합도 측정

정적 결합도는 설계 단계 시 클래스 수준에서 두 클래스들 간에 결합도를 측정하는 것으로 함수 호출의 수를 단위로 측정하게 된다.

〔정의4〕 MsgRef(f,C2) = Total # of Message passed from C1 (f) to C2

위의 식 MsgRef(f,C2)는 C1 의 입장에서 C2를 호출하는 함수의 수를 측정하게 되는데, 이는 클래스 내에 있는 하나의 함수f1이 외부 클래스인 C2의 함수들을 호출할 때, 이들 호출하는 메시지 수의 총합으로 정의한다.



〈그림 2〉 MsgRef(f1, C2)

〔정의5〕 $TM(C1,C2) = \sum_{\forall} Msgref(f_i,C_j)$

정의5 TM(Total Number of Message Reference) 에서는 C1에 포함된 전체 메소드들이 외부 클래스 C2의 메소드들을 호출하는 총합을 계산하게 된다.

〔정의6〕 $MR(C1,C2) = MAX(TM(C1,C2), TM(C1,C2))$

정의5를 이용해서는 C1에서 C2로의 일방적인 관계를 추출하기 때문에 정의6 MR(Maximum Number of Message Reference Between Two Class)을 이용하여 C1과 C2 각각의 메소드 호출의 총합 중 큰 값을 추출하고, 클래스 가중치와 동일 계층, 동일 프로젝트에 속한 클래스 관계 인지에 대한 가중치를 부여하게 된다.

정의 7에서 9까지는 가중치를 적용하기 위한 함수들의 정의로, 클래스관계 가중치, 계층 가중치, 프로젝트 가중치이다. 계층은 역할적 측면에서의 클래스 군을 정의한 것이며, 프로젝트는 기능적 측면에서의 클래스 군을 정의한 것이다.

〔정의7〕 C_Weight (C1, C2) = Weight Value for Relationship between C1 and C2

〔정의 8〕 L_Weight(C1, C2) = Weight Value for Layer Position between C1 and C2

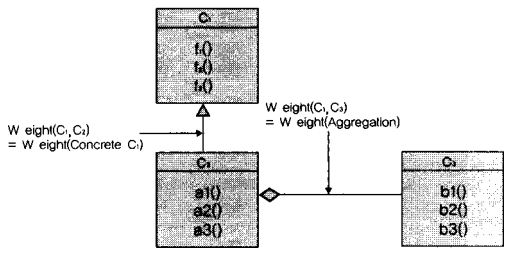
동일 계층에 존재한다는 것은 소프트웨어 개발시 소프트웨어 아키텍처상 동일한 역할을 한다는 것을 의미하며, 이는 차후 동일 컴포넌트로의 추출, 리팩토링을 통한 수퍼 클래스 추출, 통합 등의 대상이 될 수 있기 때문에 이에 대한 역할 가중치를 부여한다. 각 계층에 대한 구분을 위하여, 다음에 정의한 표 1의 지침을 따른다.

〈표 1〉 계층구분

계 층	세부역할
Service 계층	Event
	UI, view
비지니스 로직 계층	Domain-Specific Business Logic
	Common Business Logic
I/O 계층	DBMS, File, Memory, Streaming
Middleware 계층	Application Sever, TP Monitor, Middleware
Platform 계층	Network, Protocol Adaptor
	O/S

〔정의 9〕 P_Weight(C1, C2) = Weight Value for package Position Relationship between C1 and C2

패키지는 소프트웨어 시스템 개발시 공통 업무군을 구별하는 개발 단위가 된다. 일반적으로 대형 시스템의 경우에도 개별 업무에 따라 다수의 패키지 단위로 개발을 진행하므로, 이미 패키지 내에 함께 존재하는 클래스라면 클래스 간 업무적 결합도가 존재한다고 보아 이에 대한 가중치를 부여한다. 부여 방법은 동일 패키지에의 포함 여부로 결정한다.



〈그림 3〉 C_Weight(C1, C2)

다음 정의 10과 같이 MR와 3가지 가중치를 이용하여 결합도 CC(Coupling between Two Class)를 계산한다.

〔정의 10〕 $CC(C1, C2) = MR(C1, C2) \times C\_Weight(C1, C2) \times P\_Weight(C1, C2)$

3.1.3 결합도

결합도는 각 패키지 내의 메소드 중 타 패키지의 메소드를 호출하는 메소드의 결합도로 측정하며, 이 결합도는 3.1.2절에서 제시한 응집도를 그대로 사용한다.

3.2 메트릭 결과에 따른 적용 기법

단 한가지 메트릭의 기준으로의 평가할 경우 오류 발생의 위험이 있기 때문에, 제안한 복잡도, 응집도, 결합도의 통합된 결과 조건에 따라 패키지 기반 후보 컴포넌트를 추출하기 위한 기법을 아래 표2에 보이고 있다. 기호 '↑'는 패키지 또는 특정 패키지간에 적용한 메트릭이 평가한 집합 내의 평균보다 높다는 것을, 기호 '↓'는 평균보다 낮다는 것을 의미한다.

〈표 2〉 매트릭 결과 조건에 따른 컴포넌트 추출 기법

[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
1	↑	↑	↑	타 패키지와 통합
2	↑	↑	↓	하나의 컴포넌트로 추출
3	↑	↓	↑	패키지 내 클래스 간 결합도가 높은 그룹끼리 클러스터링을 통해 패키지의 분할 후 다수개의 후보 컴포넌트 추출
4	↑	↓	↓	패키지 내 클래스 간 결합도가 높은 그룹끼리 클러스터링을 통해 패키지의 분할 후 결합도가 높은 그룹을 해당 패키지로 이동
5	↓	↑	↑	패키지 자체를 결합도가 높은 타 패키지로 통합
6	↓	↑	↓	하나의 컴포넌트로 추출
7	↓	↓	↑	패키지 자체를 결합도가 높은 타 패키지로 통합
8	↓	↓	↓	하나의 컴포넌트로 추출

3.3 인터페이스 추출

인터페이스는 컴포넌트 외부의 클래스나 다른 컴포넌트가 해당 컴포넌트의 기능을 사용하고자 할 때 블랙박스 형태로 내부의 구조와 기능들이 감추어져 있는 컴포넌트에 접근할 수 있는 방법을 제공한다. 인터페이스는 컴포넌트가 가지고 있는 메소드 중 외부에서 접근 가능하도록 공개한 목록의 집합이며, 공개된 메소드가 여러 비 공개된 메소드들을 호출하는 업무 로직을 가지고 있을 수 있다. 기존 패키지 기반 객체지향 어플리케이션에서 컴포넌트의 인터페이스를 추출하기 위해, 본 논문에서는 후보 컴포넌트로 식별된 단위끼리 서로 호출하는 메소드를 인터페이스로 추출하고, 확장성을 위해 가중치를 부여한 메소드의 인터페이스를 후보 인터페이스로 둔다.

4. 평 가

본 장에서는 제안한 메트릭에 대한 이론적인 검증을 수행한다. 복잡도와 응집도가 만족해야만 하는 공리적 성질을 정의하고, 이들이 만족하는지를 검증하게 되는데 이를 위하여 Weyuker의 복잡도 성질을 이용한다. 복잡도의 이론적인 검증의 타당성을 증명하기 위해 Weyuker의 복잡도 성질을 많이 이용하는데 [17], 물론 Weyuker의 복잡도 성질이 소프트웨어 복잡도에 대한 필요충분조건은 아니지만,

현재 이론적 검증을 위해 많이 쓰이고 있을 뿐 아니라 엄격한 이론적 검증방법을 제공하고 있기 때문에, 본 논문에서도 제시한 메트릭의 타당성을 Weyuker의 성질을 이용하여 검증한다.

4.1 Weyuker의 복잡도 성질

본 논문에서 제안한 패키지의 복잡도와 응집도의 타당성을 Weyuker의 성질을 이용하여 검증한다. 다음은 Weyuker의 성질을 설명하기 위하여 사용되는 표현법이다.

P;Q는 서로 다른 임의의 클래스 P와 Q의 결합(Combination)을 나타낸다.

서로 다른 두 클래스 P와 Q가 같은 책임의 집합을 가지고 있다면, P와 Q는 동등하다고 말하고, P ≡ Q와 같이 표시한다.

μ(P)는 클래스 P의 복잡도로서 μ(P) ≥ 0 이다.

μ(P)는 클래스 P의 복잡도라고 하고, μ(Q)는 클래스 Q의 복잡도라 가정한다. 그리고 μ(P;Q)는 클래스 P와 Q를 합쳤을 때의 복잡도라 가정한다.

다음은 Weyuker의 복잡도에 대한 설명이다.

성질1 : (∃P)(∃Q) (μ(P) ≠ μ(Q))

이 성질은 모든 클래스들이 같은 척도값을 가질 수 없음을 나타내고 있다. 다시 말하자면, 서로 다른 복잡도를 갖는 클래스들이 적어도 두개는 존재한다는 것을 의미한다.

성질2 : c를 음이 아닌 수라고 하면, 복잡도가 c인 클래스 또는 프로그램의 수는 유한하다.

이 성질은 같은 복잡도를 갖는 클래스들이 유한개가 존재한다는 것을 나타내고 있다. 즉 복잡도가 c인 클래스가 주어지면, 같은 복잡도를 갖는 클래스들의 수는 유한해야만 한다는 것이다. .

성질3 : (∃P)(∃Q) (P ≠ Q & μ(P) = μ(Q))

이것은 두 개의 클래스가 같은 메트릭 값을 가질 수 있다는 것을 의미한다. 즉, 두 클래스가 같은 설계특성을 가질 수 있다는 것이다. 각 프로그램에 Unique한 수치적 이름을 할당하고, 이 이름으로 그 프로그램의 복잡도를 다루는 메트릭은 모두 이 성질을 만족하게 된다. 성질1, 2, 3은 프로그램에 직접적으로 영향을 주지 않는 것이며 메트릭으로서 만족되어야 하는 가장 기본적인 성질이다.

성질4 : (∃P)(∃Q) (P = Q & μ(P) ≠ μ(Q))

이 성질을 내포하고 있는 것은 두 개의 클래스 설계가 같은 기능을 수행할 지라도, 복잡도가 같지 않은 두 클래스가 존재한다는 것을 나타낸다. 정확하게 서로 같은 인터페이스를 제공하는 동등한 두 클래스들이 서로 다른 내부 메커니즘을 가질 수 있다.

성질5 : (∀P)(∀Q) (μ(P) ≤ μ(P;Q) & μ(Q) ≤ μ(P;Q))

이것은 두개의 클래스의 조합에 대한 메트릭은 구성요소인 클래스들 각각의 메트릭보다 결

코 작아질 수 없다는 것을 의미한다.

성질6 : $(\exists P)(\exists R)(\exists Q)\ (\mu(P) = \mu(Q)\ \&\ \mu(P;R) \neq \mu(Q;R))$

기본적으로 성질6은 같은 복잡도를 갖는 두 클래스 P와 R이 다른 클래스 Q와 각각 결합되었을 경우, 결합의 순서와 관계없이 결합된 클래스는 서로 다른 복잡도를 갖게 된다는 것을 의미하고 있다. 이것은 P와 R사이의 상호작용이 Q와 R사이의 상호작용과는 다를 수 있다는 것을 의미하는 것이다.

성질7 : 현재의 클래스 P에서 문장의 순서를 조합하여 새로 만든 프로그램을 Q라 하면, 두 클래스의 응집도나 결합도가 서로 다르다.

이 성질은 프로그램의 복잡도가 그 프로그램 내에서의 문장 순서를 반영해서 문장들 사이의 잠재적인 상호작용을 이루도록 해야 한다는 것을 의미한다.

성질8 : 클래스 P가 클래스 Q의 변수 이름을 바꾼 클래스라면, 두 클래스 P, Q의 응집도는 같다.

이 성질은 클래스의 이름을 변화시킨다 해도 클래스의 복잡도에는 영향을 주지 않아야 한다는 것을 의미한다.

성질9 : $(\exists P)(\exists Q)\ (\mu(P) + \mu(Q) < \mu(P;Q))$

이 성질은 두 개의 클래스가 결합할 때, 클래스들간의 상호작용은 메트릭 값을 증가시킬 수 있다는 것이다.

4.2 복잡도 평가

성질1 : 성질1은 3.1.1절에서 제안한 복잡도 측정 메트릭인 PCM의 값이 다른 패키지가 존재한다는 것을 나타낸다. 이 성질은 어떤 메트릭도 모든 객체지향 시스템에 대하여 같은 설계 특성을 갖지 않는다는 것을 반영하고 있으며, PCM의 값이 이를 만족한다.

성질2 : 성질2는 복잡도가 같은 패키지가 유한 개임을 나타내는데, 이것은 대부분 적용 범위가 유한 집합이기 때문에 사용되는 클래스 및 패키지의 수가 유한하다는 사실로부터 쉽게 증명된다.

성질3 : 성질3은 같은 복잡도를 갖는 서로 다른 두 패키지가 존재한다는 것을 의미하다. 즉, 두 패키지가 같은 설계 특성을 가질 수 있다는 것이다. 일반적으로 척도 정의는 수식으로 표현되기 때문에, 같은 척도 값을 갖는 패키지를 찾을 수 있다. 주어진 한 패키지 P가 가지는 책임들 가운데 한 책임의 내용을 바꾼다면, 같은 척도 값을 갖는 새로운 패키지 Q를 얻을 수 있다. 그러므로 제안된 척도는 이 성질을 만족한다.

성질4 : 성질4는 복잡도는 다르지만 같은 기능을 수행하는 패키지가 존재할 수 있다는 의미이다. 동일한 기능을 수행하지만 구현하는 방법에 따라 패키지의 복잡도가 달라질 수 있기 때문에 이 성질을 만족한다.

성질5 : 성질5는 두 패키지의 결합에 대한 복잡도가 두 요소의 패키지의 복잡도 보다 결

코 작아질 수 없다는 것을 의미한다. 일반적으로 두 패키지 P와 Q가 결합된다면 임의의 개수와 협력자의 전체 개수가 증가하게 된다. 따라서 상속되거나 또는 새로 정의되는 책임의 개수도 증가하게 된다. 만약 두 패키지가 관계를 가지고 있다면, 두 패키지의 결합은 책임의 개수가 증가함에 따라 복잡도가 증가할 것이다. 또한 관계 때문에 발생할 수 있는 새로운 기능들이 추가될 수 있고, 결합된 패키지의 협력되는 클래스의 개수도 증가하게 되므로 명백하게 복잡도는 증가하게 된다. 따라서 제안된 척도는 성질5를 만족하고 있다.

성질6 : 성질6은 같은 복잡도를 갖는 두개의 패키지에 새로운 패키지를 결합하였을 경우, 결합의 순서와 상관없이 결합된 패키지는 서로 다른 복잡도를 갖게 된다는 것을 의미한다. 복잡도가 같지만 다른 기능을 수행하는 두 개의 패키지 P, Q에 새로운 패키지 R을 각각 뒤에 첨가하였을 경우, 패키지 P와 R사이에는 같은 메소드가 존재하지만, 패키지 Q와 R 사이에는 같은 메소드가 존재하지 않을 수 있다. 이럴 경우 합해진 패키지들의 PCM값은 다를 것이므로 성질 6을 만족한다. 패키지 R을 패키지 P와 Q의 앞에 삽입했을 때에도 같은 결과를 얻을 수 있다.

성질7 : 성질7은 패키지를 구성하는 순서에 따라 복잡도가 다르다는 의미이다. 패키지의 복잡도 PCM의 값은 패키지 구성 요소들의 순서나 정의한 순서와는 아무런 상관이 없으므로 이 성질은 만족하지 않는다.

성질8 : 성질8은 패키지에서 변수의 이름을 바꾸어도 패키지 복잡도가 변하지 않는다는 의미이다. 패키지를 구성하는 클래스의 메소드나 인스턴스의 이름을 바꾸어도 패키지 복잡도는 변경되지 않기 때문에 성질 8을 만족한다.

성질9 : 성질9는 두 패키지를 결합하여 하나의 패키지로 만들면 두 패키지 각각의 PCM값보다 결합한 패키지의 PCM의 값이 더 큰 것이 존재한다는 의미이다. 패키지 P와 Q를 합하여 새로운 패키지 R을 만들었을 때, 두 개의 패키지 P와 Q가 서로 공통되는 클래스의 인스턴스와 메소드가 없다면 패키지 R의 PCM값은 P와 Q의 PCM값을 합한 것보다 더 클 수 있으므로 성질 9를 만족한다.

〈표 3〉 Weyuker의 복잡도 성질에 기준한 제안 복잡도 평가

메트릭	1	2	3	4	5	6	7	8	9
제안복잡도	○	○	○	○	○	○	×	○	○

본 연구에서 제안한 복잡도는 Weyuker의 성질7을 만족하지 못하지만, 다른 복잡도들이 만족하는 성질들을 최소한 만족하므로 복잡도로서 적합하다고 볼 수 있다.

4.3 패키지의 응집도 평가

본 절에서는 패키지의 응집도인 패키지 내의 클래스 간의 결합도에 대하여 평가한다.

성질1 : 성질1은 결합도 값이 다른 클래스가 존재할 수 있다는 것을 의미하는데, 이것이 성립하므로 이를 만족한다.

성질2 : 성질2는 클래스 결합도가 같은 클래스가 유한 개임을 나타낸다. 이것은 응집도 평가에서와 같이 대부분 적용 범위가 유한 집합이기 때문에 사용되는 참조 요소의 수와 클래스의 수가 유한하다는 사실로부터 쉽게 증명된다.

성질3 : 성질3은 결합도는 같지만 다른 기능을 수행하는 클래스가 존재할 수 있다는 의미이다. 이 성질을 증명하기 위해 서로 다른 기능을 수행하는 두 개의 클래스 P와 Q가 있다고 가정하자. 클래스P와 클래스Q가 같은 수의 외부 클래스를 참조하면서 클래스쌍의 의존도들도 같다면 각각의 CC값은 같다. 따라서 서로 다른 기능을 수행하지만 CC의 값이 같은 클래스가 존재하므로 이 성질을 만족한다.

성질4 : 성질4는 결합도는 다르지만 같은 기능을 수행하는 클래스가 존재할 수 있다는 의미이다. 이 성질을 증명하기 위해 같은 기능을 수행하는 두 개의 클래스 P와 Q가 있다고 가정하자. 클래스P는 내부에 있는 요소만을 참조하고, 클래스Q는 외부 클래스의 요소를 참조한다고 한다면 두 클래스의 결합도가 달라지게 되므로 이 성질을 만족한다.

성질5 : 성질5는 임의의 클래스에 새로운 클래스를 삽입하면 CC의 값이 단조 증가한다는 의미이다. 클래스 P와 Q를 합하여 새로운 클래스 R을 만들었을 때, 클래스 R의 CC값은 클래스 P의 값보다 작아지게 된다. 따라서 이 성질은 만족하지 못한다.

성질6 : 성질6은 같은 결합도를 갖는 두 개의 클래스에 새로운 클래스를 앞이나 뒤에 삽입할 경우 결합도가 다르다는 의미이다. 결합도가 같지만 다른 기능을 수행하는 두 개의 클래스 P와 Q에 새로운 클래스 R을 각각 뒤에 첨가하였을 경우, 클래스 Q와 R은 같은 클래스를 참조하지만, 클래스 Q와 R은 같은 클래스를 참조하지 않을 수 있다. 이럴 경우 합해진 클래스들의 CC값은 다를 것이므로 성질 6을 만족한다. 클래스 R을 클래스 P와 Q의 앞에 삽입하였을 때에도 같은 결과를 얻을 수 있다.

성질7 : 성질7은 클래스를 구성하는 순서에 따라 결합도가 다르다는 의미이다. 클래스 결합도 CC값은 클래스 구성 요소들의 구성 순서나 정의한 순서와 아무런 상관이 없으므로 이 성질은 만족하지 않는다.

성질8 : 성질8은 클래스에서 변수 이름을 바꾸어도 클래스 결합도가 변하지 않는다는 의미이다. 클래스를 구성하는 메소드나 인스턴스의 이름을 바꾸어도 클래스의 결합도는 변경되지 않기 때문에 성질 8을 만족한다.

성질9 : 성질9는 두 클래스를 결합하여 하나의 클래스로 만들면 두 클래스 각각의 CC값의 합보다 결합한 클래스의 CC값이 더 큰 것이 존재한다는 의미이다. 클래스 P와 Q를 합하여 새로운 클래스 R을 만들었을때, 두 개의 클래스 P와 Q에 있는 메소드가 같은 내부 요소를 참조하지만 다른 외부 요소를 참조하면 클래스 R의 CC값이 두개의 클래스의 CC값을 합한 것

보다 항상 작거나 같다. 따라서 성질 9를 만족하지 못한다.

〈표 3〉 Weyuker의 복잡도 성질에 기준한 제안 결합도 평가

메트릭	1	2	3	4	5	6	7	8	9
제안결합도	○	○	○	○	○	○	×	○	×

본 연구에서 제안한 결합도는 Weyuker의 성질7과 성질 9를 만족하지 못하지만 다른 결합도들이 만족하는 성질들을 최소한 만족하므로 결합도로서 적합하다고 볼 수 있다.

5. 결 론

본 논문에서는 재공학의 관점에서 최근 3년간 학계와 산업계에서 크게 주목받고 있는 컴포넌트를 추출하기 위해 메트릭을 기반으로 접근하였다. 기 개발된 객체지향 어플리케이션의 패키지 정보를 이용하여, 이를 기반으로 복잡도, 응집도, 결합도의 세가지 메트릭을 적용하고, 이들의 결과 조건에 따라 컴포넌트를 추출하는 기법을 제시하였다. 최근 한국전자통신연구원(ETRI)를 중심으로 기 작성된 코볼, 서블릿, 자바 어플리케이션을 컴포넌트화 하기 위하여 래퍼(wrapper)를 채용하려는 연구가 진행되고 있으며, 향 후 CBD와 웹 서비스(Web Services) 등 차세대 시스템을 위한 아키텍쳐의 도래, 성숙이 진행되면 기 운영 중인 어플리케이션을 폐기하고 신규 개발하거나, 이를 폐기하지 않고, 재사용 하는 사례가 늘어날 것이다. 본 논문은 객체지향의 성질을 이용하여, 컴포넌트가 객체 및 이들간의 충분한 성질을 가지고 있으며, 동일 업무 군의 범위를 벗어나지 않는 기법을 제시하여, 기 보유하고 있는 어플리케이션을 리팩토링하여 컴포넌트를 추출하는 근거와 기법을 제시하였다.

참 고 문 헌

[1] Wojtek Kozaczynski and G.Booch, " Component-based Software Engineering", IEEE Software, Vol.15, No.5, pp.34~36, September/October, 1998.

[2] Szyperski C., Component Software: Beyond Object-Oriented Programming, Addison Wesley, 1998.

[3] C.W.Krueger, "Software Reuse", ACM Computing Survey, Vol. 24, No. 2, pp.131~184, June, 1992.

[4] Hafedh Mili, Fatma Mili, Alimili, "Reusing Software : Issues and Research Direction" . IEEE Tracnsactions on Software Engineering, Vol. 21, No. 6, pp.528~562, June. 1992.

[5] Thomas M. Pigoski, Practical Software Maintenance, Wiley, pp.37~50, 1997.

[6] Roger S. Pressman, Software Engineering - A Practitioner's Approach, 4th Edition, McGraw-Hill, 1997.

[7] Souza, Desmond, Objects, Components, and Frameworks with UML, Addison Wesley, 1999.

[8] Short K., Component Based Development and Object Modeling, Sterling Software, 1997.

[9] Rational Corporation, "Rational Unified Process Version.2001A", at URL:http://www.rational.com /products/rup/tryit/eval/gen_eval.jsp.

[10] Lee, Sang Duck, Yang, Young Jong, Cho, Eun Sook, Kim Soo Dong, "COMO : A UML-Based Component Development Methodology" , Asia-Pacific Software Engineering Conference (APSEC99), Takamachu, Japan, pp.54~61, Dec.7~10, 1999.

[11] Jon Hopkins, "Component Primer", Communications of the ACM Vol.43, No.10, October2000.

[12] D. Rombach, "Design Measurement - Some Lessons Learned," IEEE Software, pp.17~25, 1990.

[13] S.R.Chidamber and C.F.Kemerer, "A Metric Suite for Object-Oriented Design" , IEEE Transactions on Software Engineering, Vol.17, No.6, pp.636~638, 1994.

[14] Lorenz, Mark and Kidd, Jeff, Object Oriented Software Metrics, Prentice Hall Publishing, 1994.

[15] T.J.McCabe, " A Complexity Measurement", IEEE Transactions on Software Engineering, , Vol.SE-2, No.4, pp.308~320, 1976.

[16] Rational Software Corp., Unified Modeling Language (UML) Summary, 1997.

[17] E.J.Weyuker, "Evaluating Software Complexity Measures" , IEEE Transactions on Software Engineering, Vol. 14, No.9, pp1357~1365, 1988.

저 자 소 개

이종호 (E-mail : jhlee@dsic.co.kr)

1985. 2. 숭실대학교 컴퓨터공학과 졸업(공학사)
1998. 2. 숭실대학교 정보과학대학원 컴퓨터공학(공학석사)
2002. 2. 숭실대학교 대학원 컴퓨터공학 (공학박사)
1994. 정보처리 기술사
1997. 정보시스템 감리인
1984. 12 ~ 1987. 6 국민은행 전산부
1987. 7 ~ 현재 대신정보통신 e-Biz 사업본부 이사
2003. 3 ~ 현재 숭실대학교 겸임교수
관심 분야 : 리엔지니어링, 분산객체컴퓨팅, 소프트웨어재사용, 소프트웨어 리팩토링, 컴포넌트시스템, 정보처리시스템감리

류성열 (E-mail : syrhew@computing.soongsil.ac.kr)

1997. 2. 아주대학교 컴퓨터학부(공학박사)
1997. 3 ~ 1998. 3. George Mason University 교환교수
1981. 3 ~ 현재 숭실대학교 정보과학대학 컴퓨터학부 교수
1998. 3 ~ 2001. 2. 숭실대학교 정보과학대학원 원장
1998. 3 ~ 현재 숭실대학교 전자계산원 원장
관심 분야 : 리엔지니어링, 소프트웨어 유지보수, 소프트웨어 재사용, 소프트웨어 재공학/역공학