컴포넌트 분석단계에 적용 가능한 컴포넌트 메트릭스

(Adaptive Component Metrics in Component Analysis Phase)

고 병 선 † 박 재 년 ††
(Byung-Sun Ko) (Jai-Nyun Park)

요 약 소프트웨어 개발의 독립성과 품질 및 생산성의 향상을 위한 재사용 기술로 컴포넌트 기반 개발(component-based development) 방법론이 사용되기 시작했다. 개발될 컴포넌트 기반 시스템의 품질 향상을 위해, 개별 컴포넌트에 대한 개발 주기의 초기 단계에 적용 가능한 컴포넌트 메트릭스에 대한 연구가 필요하다.

따라서, 본 논문에서는 컴포넌트 분석단계의 정보를 사용하여, 컴포넌트 인터페이스 복잡도(CCI)와 컴포넌트 인터페이스 응집 결여도(LCC)를 측정한다. 컴포넌트 인터페이스 복잡도는 인터페이스를 이해, 변경, 관리, 사용하기 위한 어려움을 나타낸다 할 수 있고, 컴포넌트 인터페이스 응집 결여도는 컴포넌트가 독립된 기능 단위로 내부 구성요소들이 얼마나 강하게 연관되었는지를 나타낸다 할 수 있다. 이는 시스템 개발 주기의 초기 분석 단계에서, 독립된 기능 단위로 복잡도가 낮으며 응집도가 높은 컴포넌트를 예측 가능함으로써, 최종적으로는 시스템의 품질 향상을 기대할 수 있다.

키워드 : 컴포넌트, 인터페이스, 복잡도, 응집도, 측정

Abstract The component-based development methodology becomes famous as the new way for reuse. The goal of the reuse is improvement of quality, productivity and independence on the software development. For the improvement in the quality of a component-based system, it is necessary to research component metrics in the early phase of a component development.

Hence, in this paper, we propose new component metrics using the information of a component analysis phase. Those are CCI(Complexity of Component Interface) and LCC(Lack Cohesion of Component interface). CCI indicates a difficulty about comprehension, modification, management, use of interface. LCC indicates a functional independence about how strong the elements are related with. Therefore, it is possible to predict and manage the quality of a component to be developed. Predicting a lowness of complexity and highness of cohesion as an independent functional unit by a component interface in the early phase of a component development, we can expect the improvement in the quality of a system.

Key words : component, interface, complexity, cohesion, measurement

1. 서 론

점차 소프트웨어가 대규모로 복잡하고 다양해짐에 따라, 소프트웨어의 안정성과 신뢰성을 높이기 위한 시도들이 나타났다. 이를 위해, 빠른 시간 안에 적은 비용의 높은 생산성으로 사용자의 변화하는 요구사항을 충족시키는 고품질의 시스템을 효율적으로 개발하기 위해 재사용 기술이 사용되게 되었다[1]. 또한, 유지보수를 위한 노력을 최소화하기 위한 방법으로 소프트웨어의 품질과 평가에 대한 중요성이 널리 인식되게 되었다.

† 학생회원 : 숙명여자대학교 정보과학부 컴퓨터과학전공
kobs@sookmyung.ac.kr
†† 비 회 원 : 숙명여자대학교 정보과학부 컴퓨터과학전공 교수
jnpark@sookmyung.ac.kr
논문접수 : 2002년 4월 22일
심사완료 : 2003년 2월 12일

객체지향 기술(object-oriented technology)은 실세계의 데이타와 이에 대한 처리를 캡슐화하여 클래스로 모델링하고, 이들간의 상호 관계를 줄여 재사용을 하고자 했다. 그러나, 재사용을 위해 클래스 내부의 데이타와 동작을 파악해야 하는 어려움과 재사용 단위로는 작다는 한계가 나타나기 시작하여, 그후 1990년대부터 컴포넌트 기반 개발(component-based development) 방법론이 소프트웨어 개발 방법론의 하나의 흐름으로 나타나기 시작했다. 클래스들은 그들의 상호 작용을 기반으로 하나의 서비스를 제공하기 위한 클래스들의 모임인 컴포넌트로 묶일 수 있고, 독립적인 재사용 단위인 컴포넌트를 조립함으로써 시스템을 개발하는 컴포넌트 기반 개발 방법론은 시스템 개발에서의 독립성과 재사용성 그리고 개발 시간과 비용을 줄여 생산성과 유지보수성을 향상시킬 수 있다.

독립적인 재사용 단위인 컴포넌트를 조립하여 컴포넌트 기반으로 시스템을 개발하는 경우, 새로 개발될 컴포넌트 기반 시스템의 품질은 재사용되는 개별 컴포넌트의 품질에 영향을 받는다. 따라서, 개별적으로 사용되는 컴포넌트의 폭넓은 활용과 향후 개발될 시스템의 품질 향상을 위해 개별 컴포넌트에 대한 측정은 필요하다[2, 3]. 측정(measurement)은 소프트웨어 개발 주기의 각 단계에서 산출물과 개발 절차를 제어하고 평가하거나, 소프트웨어가 포함하고 있는 속성 및 특성에 대해 정량적인 측정을 통해, 숨겨진 오류를 찾아내거나 개발할 시스템의 개발 비용이나 노력, 결함 발생률 등에 대한 예측을 가능하게 하여, 좀더 품질 좋은 시스템을 개발하도록 한다[4].

기존의 메트릭스에 대한 연구는 절차적(procedural)이거나 객체지향 소프트웨어에 적용 가능한 것이므로, 컴포넌트 기반 시스템의 개별 컴포넌트에 대한 개발 초기 단계에 적용 가능한 컴포넌트 메트릭스에 대한 연구가 필요하다. 개발 초기 단계인 분석 단계에서의 결함은 최종 산출물인 시스템의 결함으로 이어지기 때문에, 분석 단계에서의 측정은 고품질의 소프트웨어를 생산하기 위해서 반드시 수행되어야 한다[2].

따라서, 본 논문에서는 컴포넌트 분석 단계의 정보를 사용하여 개별 컴포넌트의 컴포넌트 인터페이스 복잡도와 컴포넌트 인터페이스 응집 결여도를 측정한다. 시스템 개발 주기의 초기 단계에서 보다 기능적으로 응집도가 높으며, 독립된 기능으로 복잡도가 낮아 사용이 용이한 컴포넌트를 판단 가능하며, 이로써 향후 개발될 컴포넌트의 재사용성과 생산성, 사용 용이성, 유지보수성 등을 예측하는데 사용가능 할 것이다.

2. 관련 연구

2.1 컴포넌트 기반 개발 방법론

정보 시스템 개발의 새로운 대안인 컴포넌트 기반 개발 방법론은 클래스보다 더 큰 단위인 컴포넌트를 통해 높은 추상화(abstraction)와 재사용성(reuse)을 목표로 하는 개발 방법론이다. 컴포넌트 기반 개발 방법론은 전통적인 개발 방법론의 연장선으로, 소프트웨어도 하드웨어 부품처럼 조립이 가능해 개발자가 필요로 하는 컴포넌트를 구입해 설치만 하면 동작이 가능한 시스템을 구성할 수 있다는 것으로, 이를 통해 개발 시간과 비용을 줄이고 생산성을 향상시킬 수 있다. 또 다른 효과로 독립적인 컴포넌트의 조립으로 유지보수가 용이해짐에 따라 고품질의 소프트웨어 개발을 기대할 수 있다[3, 4, 5, 6, 7].

컴포넌트 기반 개발 방법론의 개발 절차는 (그림 1)처럼 객체지향 개발 방법론과 동일하게 반복적(iterative)이며 점진적(incremental)인 특성을 따르는 개발 절차를 따른다. 그림의 가로축인 개발 단계(phase)는 거시적으로 개념(inception) 단계, 정제(elaboration) 단계, 구축(construction) 단계, 전이(transition) 단계의 점진적인 4단계로 구성된다. 그리고, 그림의 세로축은 개별 컴포넌트를 생성하는 처리과정(process components)과 컴포넌트의 조립으로 시스템을 구축하는 구성과정(supporting components)으로 이루어진다. 처리과정에 대한 미시적 단계는 요구사항(requirement) 정의 단계, 컴포넌트 명세(specification)를 작성하는 분석 및 설계(analysis & design) 단계, 컴포넌트를 구현하는 구현(implementation) 단계, 그리고 테스트(test) 단계로 개별 컴포넌트에 대한 과정이다. 구성과정에 대한 미시적 단계는 개별 컴포넌트의 조립으로 시스템을 개발하기 위한 변화되는 환경을 관리하는 관리(management) 단계, 환경(environment) 설정 단계, 배치(deployment)

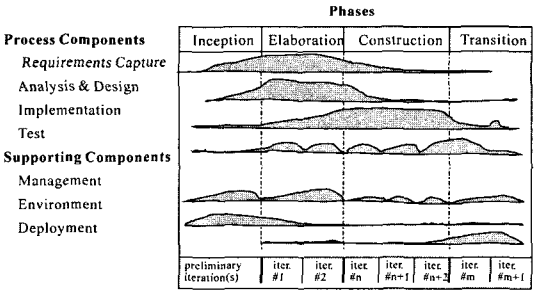

그림 1 컴포넌트 기반 개발 절차

단계로 이루어진다[8].

2.2 컴포넌트 기반 시스템의 컴포넌트 분석 및 설계

컴포넌트 기반 개발 방법론은 여러 가지가 있으나, 아직 표준화된 컴포넌트 모델링 방법론은 없다. 그러나, 시스템의 정적 및 동적 기능을 분석하는 문제영역 이해 단계, 얻어진 정보로 독립적인 컴포넌트의 구조를 설계하는 아키텍처(architecture) 정의 단계, 컴포넌트 구현과 공급을 통한 문제 해결 단계의 기본적인 세 단계로 구성된다는 점은 공통이다.

본 절에서는 앞 절에서 살펴본 컴포넌트 기반 시스템의 개발 절차 중 개별 컴포넌트를 생성하는 컴포넌트 처리 과정에 대한 개발 절차에 대해서만 살펴보겠다[9].

요구사항 분석 단계에서는 업무 영역의 요구사항을 분석하여 기능 중심의 유즈케이스 모델(use case model)을 통해 액터(actor)와 시스템이 제공하는 기능을 파악한다.

분석 단계에서는 컴포넌트 구조 설계(component architecture design)과 인터페이스 상호작용 모델링(interface interaction modeling)이 이루어진다. 객체지향 개발의 분석 단계에서는 클래스가 도출되지만, 컴포넌트의 분석 단계에서는 컴포넌트와 그들의 인터페이스가 도출되는 차이점을 갖는다. 컴포넌트 구조 설계는 요구사항의 분석 결과를 토대로 업무 영역의 클래스들을 서로 연관되고 의존하는 것끼리 중요한 클래스를 중심으로 높은 응집력을 갖도록 컴포넌트로 분할하고, 각 컴포넌트들이 갖게 될 인터페이스를 정의한다. 그리고, 인터페이스 상호작용 모델링은 시스템의 행위를 분석한다. 이렇게 시스템의 구조와 행위를 파악하고 나면, 컴포넌트를 구성하고 있는 내부 클래스들을 파악하여 클래스들의 관계를 정의하고 컴포넌트가 제공하는 기능들을 정의하는 인터페이스 정의(interface definition)를 수행하여, 컴포넌트 구조 설계를 완성한다.

설계 단계에서는 컴포넌트의 구현 환경에 대한 고려를 통해, 분석 단계의 모델에 대해 구현 환경을 고려한 상세 설계를 한다.

구현 단계에서는 구현 환경에 적합한 컴포넌트를 구현하게 되며, 테스트 단계에서는 구현된 컴포넌트에 대해 요구사항을 만족시키는 컴포넌트인지를 시험하게 된다.

이러한 개별 컴포넌트의 개발 절차는 객체지향 개발 방법론과 매우 유사함을 알 수 있다. 구현 환경에 상관없이 시스템이 요구하는 기능을 파악하여 분석 모델을 만들고, 구현 환경을 반영하기 위해 설계 모델을 만들고, 이를 구현하고 시험하는 과정은 유사하다.

2.3 컴포넌트의 정의 및 특징

객체지향 기술은 실세계의 데이타와 이에 대한 처리를 캡슐화하여 클래스로 모델링하고, 이들간의 상호 관계를 최소로 줄여 재사용을 하고자 했다. 그러나, 이를 위해서는 클래스의 내부 코드를 파악해야 하는 어려움이 있어 클래스 재사용에 한계가 나타났다. 따라서, 객체지향 시스템의 클래스들은 그들의 상호 작용을 기반으로 하나의 기능을 제공하기 위한 클래스들의 모임인 개별 특성을 갖는 컴포넌트로 묶일 수 있다.

컴포넌트는 업무의 기능과 관련된 데이타를 하나의 단위로 취급한 독립적으로 개발 보급되는 구조화된 소프트웨어 단위이다. 일반적으로, 객체지향 시스템의 클래스들은 독립된 기능 단위인 컴포넌트로 묶이게 되고, 컴포넌트가 제공하는 서비스는 인터페이스로 표현되는데, 이는 컴포넌트 내부의 클래스들에 의해 수행되는 서비스에 접근하기 위한 행위(operation)의 집합이다[9]. 다시 말하면, 독립적 기능 단위인 컴포넌트는 상태를 나타내는 데이타와 그 데이타를 처리하는 업무로써, 내부의 클래스와 외부의 인터페이스로 구성되는 기능 독립성을 갖는다. 외부에서 컴포넌트의 서비스에 접근하기 위해서는 인터페이스를 이용하며, 인터페이스는 컴포넌트 내부의 클래스들 사이에 메시지를 통해 데이타를 교환하며 상호 작용을 함으로써 컴포넌트의 서비스를 제공하는데[12, 14], 이는 클래스들이 하나의 컴포넌트로 통합되기 위한 응집성을 높인다[2, 6, 10, 11]. 그렇지만, 인터페이스는 컴포넌트가 어떤 방법으로 서비스를 제공하는지의 컴포넌트 내부를 보이지는 않는데, 이는 객체지향 시스템의 특성 중 캡슐화(encapsulation)와 정보은닉(information hiding) 개념으로 대표되는 추상화에 대응된다.

컴포넌트에 대해서는 관점에 따라 여러 가지 정의들이 가능하다. 객체 지향의 관점에서는 객체의 집합으로, 서비스의 관점에서는 독립적으로 사용할 수 있는 서비스 그룹으로 설명하기도 한다. 본 논문에서는 분석 단계의 컴포넌트에 대해 초점을 맞추며, 컴포넌트를 독립적인 소프트웨어 모듈로 특정 서비스를 제공하기 위한 클래스들의 묶음으로, 외부에서 컴포넌트 내부의 클래스들의 서비스에 접근하기 위해서는 인터페이스를 이용한다고 정의하겠다.

컴포넌트의 대표적인 특징은 재사용이 가능한 서비스 패키지라는 것이다. 이는 컴포넌트의 목적을 가장 잘 나타내는 것으로, 재사용을 통하여 소프트웨어 개발 비용 및 시간을 줄이고 또한 품질 향상을 도모하고자 하는 것이다. 컴포넌트는 잘 정의된 인터페이스를 통하여 다

른 컴포넌트와 상호 연계되는 블랙박스(black box)와 같은 것이며, 내부의 구현이 캡슐화되어 인터페이스와 완전히 분리되어 컴포넌트간의 상호 연계를 쉽게 하는 특징을 갖는다. 또한, 병행 개발될 수 있고, 분산 환경에 적합하다는 특징을 갖는다[12, 13].

2.4 컴포넌트 메트릭스의 필요성

객체지향 시스템은 속성과 메소드가 캡슐화된 클래스를 기반으로 개발된다. 그러므로, 대부분의 객체지향 메트릭스는 객체지향 시스템의 기본 단위인 클래스에 기반을 두고, 클래스, 메소드, 상속성, 캡슐화 등을 고려하여 품질을 측정하였다[14].

그러나, 컴포넌트 기반 시스템은 하나 이상의 컴포넌트로 이루어지며, 또한 컴포넌트는 하나 이상의 클래스와 인터페이스로 구성되는 클래스와는 다른 구조적 특성을 가지므로, 객체지향 메트릭스를 그대로 컴포넌트에 적용하는 것은 부적당하다.

클래스는 재사용의 단위로서 너무 작은 기능 단위일 뿐만 아니라, 다른 클래스들에 의존하는 경우가 많기 때문에 개별적으로 재사용 되기 어렵다. 따라서, 서로 연관되고 의존하는 클래스들을 모아 하나의 컴포넌트를 구성하고, 이러한 컴포넌트의 재사용을 통해 소프트웨어 개발에 대한 재사용성, 유지보수성의 향상 효과를 기대할 수 있다. 개별 컴포넌트가 제공하는 서비스는 인터페이스를 통해서 정의되며, 구체적으로 인터페이스의 서비스 수행은 여러 오퍼레이션으로 제공된다.

그러므로, 상호작용이 이루어지는 컴포넌트 내부의 구성요소인 클래스들과 컴포넌트의 서비스에 대한 표현인 인터페이스와 같은 컴포넌트에 대한 추가적인 정보를 고려한, 컴포넌트 기반 시스템의 개별 컴포넌트에 대한 개발 초기 단계에 적용 가능한 새로운 컴포넌트 메트릭스가 필요하다.

3. 컴포넌트 메트릭스

3.1 개요

컴포넌트 기반 시스템의 컴포넌트는 클래스들의 상호작용을 기반으로 하나의 기능을 제공하기 위한 클래스보다 큰 재사용을 위한 독립적인 소프트웨어 단위이다. 컴포넌트 분석 단계에서 식별된 특정 기능을 수행하는 컴포넌트는 컴포넌트 기반 시스템의 구현된 컴포넌트로 대응되어 재사용 가능하다. 만일, 컴포넌트가 특정 영역의 요구사항을 부적절하게 파악하여, 분석 단계에서 식별된 컴포넌트가 만족스럽지 못하다면, 향후 컴포넌트의 구현 및 구현된 컴포넌트의 재사용에 좋지 않은 결과를 나타내게 될 것이다. 그러므로, 시스템 개발 주기의 초기 단계에서 컴포넌트가 적절하게 구성되었는지를 측정하는 일은 매우 중요하며 필요하다.

객체지향 시스템의 클래스는 하나의 기능 제공을 위한 속성과 메소드의 집합이라는 캡슐화의 특성을 만족한다. 그러나, 컴포넌트에서는 명세와 기능의 분리라는 완벽한 캡슐화의 특징을 만족하며, 객체지향 언어뿐만 아니라 어떤 프로그래밍 언어로도 구현이 가능하다. 객체 지향 기술이 컴포넌트 기반 개발의 필요충분 조건은 아니지만, 컴포넌트 기반 개발을 지원하기에 가장 적합한 개발 기술이다. 그러므로, 컴포넌트가 객체 지향적 특성인 상속성, 복잡도, 응집도, 결합도의 측면에서 얼마나 적절히 설계되었는가를 측정할 수 있을 것이다. 컴포넌트는 클래스와 달리 상속을 통한 재사용이 허용되지 않으므로 상속성은 측정 분야에서 제외하며, 본 논문은 분석 단계의 개별 컴포넌트에 대해서만 관심을 두므로, 컴포넌트간의 결합도는 측정 분야에서 제외하고, 복잡도와 응집도 측면에서 컴포넌트 설계의 적절성을 판단하고자 한다. 복잡도는 소프트웨어의 변경이나 이해 정도의 어려움을 측정하는 것으로, 소프트웨어의 신뢰성, 이해성, 유지 보수성, 변경 용이성, 테스트 용이성과 같은 품질 특성에 영향을 준다[10]. 응집도는 소프트웨어 구성요소들 사이의 연관 정도를 측정하는 것인데, 응집도가 높을수록 모듈의 독립성은 높아져 복잡도가 감소되고, 또한 신뢰성, 이해성, 유지 보수성과 같은 품질 특성에 영향을 준다[11]. 컴포넌트의 사용을 위해 외부에서 접근하는 인터페이스의 복잡도가 높지 않다면, 인터페이스의 사용이 쉬워져 컴포넌트의 사용 용이성이 좋아질 것이며, 인터페이스와 컴포넌트 내부의 클래스들간의 협력의 복잡도가 낮을수록 컴포넌트의 유지보수성이 좋아질 것이다. 또한, 독립된 기능 단위인 컴포넌트 내부 구성요소들 사이의 응집성이 높아야 컴포넌트의 기능 독립성과 재사용성, 유지 보수성이 좋아질 것이다.

본 논문에서 제안하는 컴포넌트 메트릭스는 컴포넌트의 사용 용이성, 재사용성 및 유지 보수성 향상을 위한 컴포넌트 인터페이스 복잡 정도와 기능 독립성을 위한 컴포넌트 인터페이스 응집 정도에 기반한다. 컴포넌트 기반 시스템의 임의의 한 컴포넌트 Co에 대해 이루어지는 컴포넌트 품질 측정은 $E(Co)=\{CCI, LCC\}$ 로 표현 가능하며, 컴포넌트 인터페이스 복잡도 CCI(Complexity of Component Interface)와 컴포넌트 인터페이스 응집 결여도 LCC(Lack of Cohesion in Component) 메트릭스로 이루어진다.

3.2 컴포넌트 인터페이스 복잡도

컴포넌트 인터페이스는 컴포넌트 내부의 클래스들에

의해 수행되는 서비스에 대한 축약 표현이 되며, 또한 컴포넌트 내부의 여러 클래스들에 공통적으로 접근하기 위한 통로 역할을 하게 된다[2, 9]. 독립적 기능 단위인 컴포넌트가 특정 서비스를 제공하기 위해 인터페이스는 기능적으로 컴포넌트 내부의 클래스들과 협력 관계를 갖게 된다. 인터페이스의 이러한 협력 관계는 인터페이스의 오퍼레이션(operation)을 통해 제공되며, 구체적으로는 일종의 메소드인 오퍼레이션이 매개변수를 통해 클래스들과 협력 관계를 갖는다.

컴포넌트의 사용 용이성과 재사용성을 위해서는 외부에서 접근하여 사용하는 컴포넌트 인터페이스의 복잡 여부가 컴포넌트의 사용 및 유지보수를 결정하게 될 것이다. 따라서, 컴포넌트 인터페이스 복잡도는 서비스를 제공하기 위한 컴포넌트의 인터페이스를 이해 및 개발하기 위한 총체적인 어려움을 나타내며, 오퍼레이션의 수로 측정한다.

컴포넌트 인터페이스 복잡도 CCI(Complexity of Component Interface)는 다음과 같다.

$$CCI(Co) = \sum_{k=1}^{o} |Op_k|$$

여기서, Op는 인터페이스의 오퍼레이션이다.

컴포넌트 인터페이스 복잡도는 오퍼레이션의 수가 많은 인터페이스는 기능 수행 과정 및 컴포넌트 사용 및 이해가 복잡하여 어렵다는 것을 나타낸다. 측정값은 인터페이스의 복잡도를 나타내므로 너무 크다면 좋지 않다.

3.2.1 크기 메트릭 속성에 대한 평가

컴포넌트 인터페이스 복잡도는 인터페이스의 크기에 대한 메트릭이다. 메트릭의 이론적 평가는 Briand[15]가 제안한 크기(size) 메트릭이 만족해야만 하는 필요조건적 속성에 적용하여 평가해 본다.

성질 1-1. Non-negativity

측정값은 음수(negativity)의 값을 가질 수 없다는 것을 의미한다. 인터페이스는 서비스 수행을 위한 오퍼레이션으로 구성되는데, 인터페이스를 구성하는 오퍼레이션이 없다면 최악의 경우 0일 수는 있으나, 음수는 아니므로, 성질 1-1은 만족된다.

성질 1-2. Null Value

측정값은 0 일 수 있다는 것을 의미한다. 성질 1-2에서도 설명되었듯이, 인터페이스를 구성하는 오퍼레이션이 없다면, 측정값은 0이므로, 성질 1-2는 만족된다.

성질 1-3. Module Additivity

어느 컴포넌트의 인터페이스가 서로 공통 부분이 없는 두 그룹으로 분할 가능하다면, 인터페이스는 두 그룹의 합집합과 같다는 것을 의미한다. 인터페이스는 컴포넌트 서비스 수행을 위한 여러 오퍼레이션으로 구성되고, 그들 사이에 중복이 없는 두 그룹으로 분할된다면, 인터페이스의 전체 크기는 오퍼레이션들의 두 그룹의 합과 동일하므로, 성질 1-3은 만족된다.

3.2.2 컴포넌트 크기에 대한 증명

컴포넌트는 특정 서비스를 제공하는 기능 단위인데, 설계자들은 너무 커서 다루기 어려운 컴포넌트나 재사용을 하기에 너무 작은 컴포넌트의 설계는 좋지 않다고 생각한다. 즉, 컴포넌트는 기능의 단위로 너무 크지도 또한 너무 작지도 않아야 한다.

따라서, 컴포넌트 인터페이스 복잡도는 인터페이스의 크기에 대한 메트릭이므로, 제안된 컴포넌트 인터페이스 복잡도 측정이 컴포넌트 결합과 분할에 대해 어떻게 반응하는 가에 대해 증명한다.

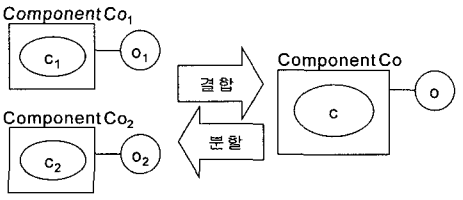

그림 2 컴포넌트의 결합과 분할

첫째, 컴포넌트 결합에 대해 살펴본다. (그림 2)와 같이 o_1과 o_2개의 오퍼레이션으로 구성되는 인터페이스와 c_1과 c_2개 클래스로 구성되는 두 컴포넌트 Co_1과 Co_2가 새로운 컴포넌트 Co로 결합되었다고 하자. 새로운 인터페이스는 o_1과 o_2개의 오퍼레이션을 결합한 것인데, 이때 행위들 사이에 중복이 발생할 수 있으므로, $o_1 + o_2 \geq o$이 된다. 또한, 결합되는 클래스들 사이에도 중복이 발생할 수 있으므로, $c_1 + c_2 \geq c$이 된다. 이때, 컴포넌트 인터페이스 복잡도 계산은 아래와 같으며, 서로 비슷한 기능을 하는 두 컴포넌트를 하나의 컴포넌트로 결합할 경우 좀더 낮은 복잡도를 갖게 된다는 것을 나타낸다.

$$\begin{aligned} CCI(Co_1) + CCI(Co_2) &= o_1 + o_2 \\ &\geq o \\ &= CCI(Co) \end{aligned}$$

둘째, 컴포넌트 분할에 대해 살펴본다. (그림 2)와 같이 c개의 클래스로 구성되고, 인터페이스의 전체 오퍼레이션의 수는 o개인 컴포넌트 Co가 c_1과 c_2개의 클래스와 o_1과 o_2개의 오퍼레이션으로 구성되는 인터페이스를 갖는 두 컴포넌트 Co_1과 Co_2로 분할되었다고 하자. 컴포넌트의 분할은 보다 작은 기능 단위로의 분할을 의미

하므로, 기능의 축소로 기본 행위의 수보다 감소할 수 있으므로, $o \geq o_1 + o_2$ 이다. 이때, 컴포넌트 인터페이스 복잡도 계산은 아래와 같으며, 이는 규모가 큰 하나의 컴포넌트가 적당한 크기의 두 컴포넌트로 분할될 경우 좀더 낮은 복잡도를 갖게 된다는 것을 나타낸다.

$$\begin{aligned} CCI(Co) &= o \\ &\geq o_1 + o_2 \\ &= CCI(Co_1) + CCI(Co_2) \end{aligned}$$

3.3 컴포넌트 인터페이스 응집 결여도

응집도(cohesion)는 모듈 내부 구성 요소들이 서로 얼마나 관련되어 있는지의 정도를 나타내는 소프트웨어 속성(attribute)으로, 모듈의 기능 독립성(functional independence)을 측정할 수 있다. 소프트웨어 요소들이 서로 밀접하게 상호작용을 한다면 같은 모듈안에 포함되어야 하며, 응집도가 높은 모듈은 하나의 기능만을 수행하여 여러 개로 나누기 어려우며, 소프트웨어의 설계와 유지보수가 쉬워진다는 특성을 갖는다[9, 15].

응집도는 한 모듈 내부의 기능적인 연관 정도를 나타내는 척도로, 내부의 처리 요소들간의 상호 관련 정도에 따라 다음과 같이 7가지 종류로 분류된다[6]. 응집도가 높을수록 모듈 내부 구성요소들의 기능적인 연관 정도가 강한 경우로, 기능적 응집도가 연관 정도가 가장 높고, 우연적 응집도가 가장 낮다.

① 기능적(functional) 응집도 : 모듈 내부의 모든 요소들이 한 가지 기능을 수행하기 위해 구성될 때, 이 모듈은 기능적 응집도를 갖는다고 한다.

② 순차적(sequential) 응집도 : 실행되는 순서가 서로 밀접한 관계를 갖는 기능을 모아 한 모듈로 구성한 경우이다. 한 기능 요소에 의한 데이타가 다음 기능 요소의 입력 데이타로 제공되는 경우로, 이 모듈은 순차적 응집도를 가진다고 한다.

③ 교환적(communicational) 응집도 : 모듈 내부의 각 요소 사이에서 데이타를 주고받거나 데이타를 공유하는 경우 교환적 응집도를 갖는다고 한다.

④ 절차적(procedural) 응집도 : 어떤 모듈이 다수의 관련 기능을 순차적으로 수행하는 경우 절차적 응집도를 갖는다고 한다.

⑤ 시간적(temporal) 응집도 : 기능들 사이에 강한 관련성은 없으며, 어느 특정한 시간에 처리되는 몇 개의 기능을 모아 하나의 모듈을 구성한 경우 시간적 응집도를 갖는다고 한다.

⑥ 논리적(logical) 응집도 : 논리적으로 서로 관련이 있는 요소들을 모아 하나의 모듈로 구성한 경우 논리적 응집도를 갖는다고 한다.

⑦ 우연적(coincidental) 응집도 : 모듈 내부의 각 요소들이 서로 특별한 관계가 없는 경우 우연적 응집도를 갖는다고 한다.

함수나 프로시저로 정의되는 절차적 시스템의 응집도 개념은 객체지향 시스템의 클래스에 의해 확장되어 정의될 수 있다. 이는 객체지향 시스템의 특성 중 추상화(abstraction)의 개념과 대응되며, 추상화는 캡슐화(encapsulation)와 정보 은닉(information hiding) 개념으로 표현되기도 한다. 또한, 클래스 응집도는 클래스들의 묶음인 컴포넌트 응집도로 발전시킬 수 있다.

특정 기능을 수행하는 독립된 기능 단위인 컴포넌트는 서비스 제공을 위해 필요한 요소인 데이타와 동작으로써 클래스와 인터페이스를 갖는다. 클래스보다 규모가 큰 재사용 단위인 컴포넌트는 서로 연관되고 의존하는 클래스들로 구성되며, 외부에서 컴포넌트에 접근하기 위한 매개자 역할을 하는 인터페이스는 컴포넌트 내부의 클래스들과의 상호작용을 통해 컴포넌트 서비스를 제공한다. 컴포넌트 모델링이 잘 되었다면, 컴포넌트 구성 요소인 클래스들 사이에 밀접한 관련성을 갖는 응집도가 높은 컴포넌트가 될 것이다. 좋은 컴포넌트는 기능 수행을 위해 내부 구성 요소들간에 높은 응집력(highly cohesive)을 가져야 하며, 외부의 다른 컴포넌트와는 작은 결합력(loosely coupled)을 가져야 한다.

따라서, 시스템의 설계 품질을 측정하기 위한 대표적 척도인 응집도를 이용해 컴포넌트의 기능 독립성을 파악할 수 있다. 컴포넌트는 서비스를 제공하기 위해 인터페이스와 내부 클래스들이 기능적으로 상호작용을 하는데, 오퍼레이션이 컴포넌트 내부의 클래스들을 매개변수로 사용하는 비율로 인터페이스에 의한 컴포넌트 응집도를 측정할 수 있고, 이는 컴포넌트의 기능 독립성을 나타낸다. 그러나, 본 논문에서는 컴포넌트의 설계를 방해하는 요소 측면에서 컴포넌트 응집 결여 정도를 측정하여, 컴포넌트의 설계가 잘못 되었음을 판단하고자 한다.

컴포넌트 인터페이스 응집 결여도 LCC(Lack of Cohesion in Component interface)는 다음과 같다.

$$LCC(Co) = 1 - \frac{\sum_{k=1}^{n} |P_C(Op_k)|}{\sum_{k=1}^{n} |P(Op_k)|}$$

여기서,

$P(Op_k) = \{p \mid p$는 Op_k의 *parameter*$\}$ 이며,

$P_C(Op_k) = \{p_c \mid p_c$는 Op_k의 *class type*인 *parameter*$\}$ 이다.

컴포넌트 인터페이스 응집 결여도는 인터페이스의 오

퍼레이션이 매개변수로 컴포넌트 내부의 클래스를 공통적으로 사용하지 않아, 컴포넌트가 서비스 수행을 위한 내부 구성요소들의 기능적 연관 정도가 어느 정도인지를 측정한다.

컴포넌트 인터페이스가 서비스를 제공하기 위해 내부 구성요소들과 얼마나 기능적으로 연관되는지를 측정하기 위해, 수식은 인터페이스를 구성하는 오퍼레이션의 전체 매개변수의 수에 대해 컴포넌트 내부의 클래스를 매개변수로 사용하는 클래스 타입인 매개변수의 수를 비율(ratio)로 구한다. 이는 컴포넌트 인터페이스의 기능 응집도라 할 수 있으며, 컴포넌트의 크기인 인터페이스의 오퍼레이션의 수나 내부 클래스의 수에 영향을 받지 않도록 0에서 1사이의 값으로 정규화가 되었고, 오퍼레이션의 전체 매개변수의 수를 고려하므로 컴포넌트 내부 구성 요소들의 전체적 관계를 모두 고려하게 된다. 그러나, 본 논문의 컴포넌트 인터페이스 응집 결여도는 컴포넌트 설계가 적절하지 못함을 판단하기 위한 것으로, 계산된 응집도 값을 최대 응집도인 1과의 차이로 계산하므로, 일반적인 응집도와 달리 값이 클수록 응집력이 약한 경우이며, 값이 작을수록 응집력이 강한 컴포넌트가 되어 바람직한 경우이다.

3.3.1 응집도 메트릭 속성에 대한 평가

컴포넌트 인터페이스 응집 결여도는 응집도에 대한 메트릭이다. 응집 결여도는 계산된 응집도 값을 최대 응집도인 1과의 차이로 계산하므로, 일반적인 응집도와 달리 그 값이 작을수록 응집도가 좋아진다는 의미를 갖는다는 점만 다르다. 그러므로, 컴포넌트 인터페이스 응집 결여도 메트릭의 이론적 평가는 Briand[15]가 제안한 응집도 측정이 가져야만 하는 필요조건적 성질과 Weyuker[16]의 성질 중 Merha[11]가 재정의한 응집도 관련 성질에 적용하여 컴포넌트 인터페이스 응집도를 평가해 본다.

성질 2-1. Non-negativity and Normalization

측정값은 음수의 값을 가져서는 안되며, 항상 일정한 구간내에 존재해야 한다는 것을 의미한다. 구성 요소들 사이에 최대의 관련성을 갖는다면, 응집도는 모듈의 크기에 비례하는 최대값을 갖는다. 그러나, 응집도가 모듈의 크기에 영향을 받지 않도록 하기 위해, [min, max] 구간의 값으로 정규화(normalization)를 시킬 수 있고, 음수(negativity)의 값을 가질 수 없으므로, 성질 2-1은 만족한다.

성질 2-2. Null Value

특정 모듈의 내부 구성요소들 사이에 연관성이 전혀 없다면, 응집도는 0일 수 있다는 것을 의미한다. 응집도는 구성요소들 사이의 연관 정도를 나타내는 값이므로, 모듈의 구성 요소들 사이에 전혀 관련이 없는 최악의 경우라면 응집도는 0이므로, 성질 2-2는 만족된다.

성질 2-3. Monotonicity

모듈의 내부 구성요소의 양적 변화 없이 구성요소들 사이의 연관성만 증가한다면, 모듈의 구성요소들이 하나의 모듈로 통합(encapsulation)되기 위한 요소의 증가를 의미하므로, 응집도는 증가하게 된다는 것을 의미한다. 만일, 컴포넌트의 인터페이스나 내부 클래스의 양적 변화 없이, 인터페이스가 내부 클래스를 사용하는 정도만 증가한다면, 구성요소들 사이의 연관 정도가 증가하는 것이므로, 응집도는 증가하지 감소하지는 않는다. 따라서, 성질 2-3은 만족한다.

성질 2-4. Cohesive Modules

서로 관련이 없는 두 모듈을 합쳤을 경우, 응집도는 원래 모듈의 최대 응집도보다 증가하지 않는다는 것을 의미한다. 서로 관련이 없는 두 컴포넌트를 하나로 통합할 경우, 관련없는 요소들이 함께 통합되어 컴포넌트의 구성 요소들 사이에 분리(dividing) 집합이 생기게 되고, 이는 구성요소들이 하나의 모듈로 통합되기 위한 관계의 감소를 의미한다. 이런 통합된 컴포넌트에서 인터페이스가 컴포넌트 내부의 클래스를 사용하는 경우는 증가하지 않으므로, 구성요소들의 응집력은 낮아지고, 새로운 컴포넌트의 응집도는 원래 두 컴포넌트의 최대 응집도보다 크지 않다. 따라서, 성질 2-4는 만족한다.

성질 2-5. Renaming

컴포넌트가 그 이름을 바꾼다해도 응집도에는 변함이 없다는 것을 의미한다. 컴포넌트의 이름이나 그 내부 클래스의 이름에 기반을 두지 않기 때문에, 응집도에는 변함이 없다는 것이 명백하다. 따라서, 제안한 컴포넌트 응집도는 이름에 영향을 받지 않으므로, 성질 2-5는 만족한다.

성질 2-6. Transitive

서로 다른 응집도 값 사이에, "$A<B$이고 $B<C$이면, $A<C$"라는 이행적 성질을 만족한다는 것을 의미한다. 응집도는 구성 요소들 사이의 관련 정도를 수치화한 것으로 크기에 영향을 받지 않도록 정규화 하게 된다. 따라서, 정규화된 응집도는 서로 다른 모듈들 간에 값의 상대적 비교가 가능하므로, 값의 크기에 대해 이행적 성질을 만족하므로, 성질 2-6은 만족된다.

성질 2-7. Coarse

다른 응집도 값을 갖는 서로 다른 모듈들이 존재한다는 것을 의미한다. 한 시스템은 여러 컴포넌트로 구성되고, 한 컴포넌트는 인터페이스와 내부의 클래스들의 상

호작용으로 구성된다. 따라서, 같은 인터페이스와 같은 클래스들로 구성되며, 같은 상호작용이 관계를 이루며 구성되는 컴포넌트는 하나이상 존재할 수 없다. 그러므로, 서로 다른 두 컴포넌트로부터 계산된 응집도가 다르다는 사실을 쉽게 알 수 있다. 따라서, 성질 2-7은 만족한다.

4. 사례 연구

사례 연구는 컴포넌트 기반 비디오 관리 시스템으로, 비디오 대여를 위한 Rental Manager 컴포넌트, 비디오 예약을 위한 Reservation Manager 컴포넌트, 고객 관리를 위한 Membership Manager 컴포넌트의 세 컴포넌트로 구성된다.

(그림 3)은 시스템의 Reservation Manager 컴포넌트로, 이를 구성하는 클래스와 오퍼레이션의 집합인 인터페이스를 표현한 것이다. 이에 본 논문에서 제안한 컴포넌트 인터페이스 복잡도와 컴포넌트 인터페이스 응집 결여도를 적용한 측정 결과는 <표 1>과 같다. 측정 결과로 종합적으로 판단해 볼 때, Reservation Manager 컴포넌트가 가장 컴포넌트 인터페이스의 복잡성이 낮으며, 응집성이 높음을 알 수 있다. 메트릭스의 측정값으로부터 분석 단계의 컴포넌트에 대해 인터페이스의 복잡성과 응집성을 판단 가능하다. 일반적으로, 복잡성이 낮으며 응집성이 높은 모듈은 사용 및 유지보수가 용이하므로, 이로써 컴포넌트 개발을 위한 노력 및 비용을 예측 가능하여, 컴포넌트 분석 모델의 적절성을 예측 및 판단하는 용도로 메트릭스를 사용 가능하다.

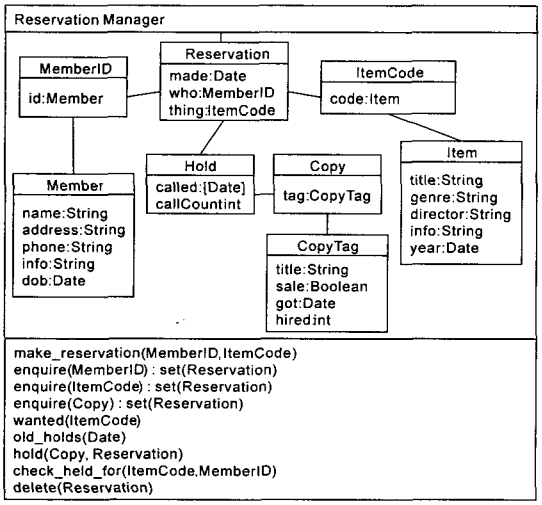

그림 3 Reservation Manager 컴포넌트의 구성

표 1 사례 연구의 측정값

컴포넌트 / 측정	Rental Manager	Reservation Manager	Membership Manager
CCI	12	9	6
LCC	0.38	0.08	0.44

5. 결 론

컴포넌트 분석단계의 정보를 사용하여 컴포넌트 인터페이스가 서비스를 제공하기 위해 얼마나 복잡하며, 독립된 기능 단위로 얼마나 응집되었는지를 측정하였다. 컴포넌트 인터페이스 복잡도는 인터페이스를 이해, 변경, 관리하기 위한 어려움을 나타낸다 할 수 있고, 컴포넌트 인터페이스 응집 결여도는 컴포넌트가 독립된 기능 단위로 내부 구성요소들이 얼마나 강하게 연관되었는지를 나타낸다 할 수 있다. 개별 컴포넌트의 개발을 위한 컴포넌트 분석 단계에서 본 논문에서 제안한 컴포넌트 메트릭스를 사용하면, 시스템 개발 주기의 초기 단계에서 보다 기능적으로 응집도가 높으며, 독립된 기능으로 복잡도가 낮은 컴포넌트를 판단 가능하며, 이로써 향후 개발될 컴포넌트의 재사용성, 생산성, 유지보수성 등을 예측 가능하다. 다시 말하면, 컴포넌트 인터페이스에 대한 복잡성과 응집성의 측정은 개발 비용, 노력 및 시간, 결함 발생률 등을 예측하는데 사용 가능하며, 이로써 컴포넌트의 재사용성, 유지보수성, 기능 독립성, 신뢰성, 이해성, 생산성과 같은 품질 특성 측면에서 컴포넌트를 평가하기 위해 사용될 수도 있을 것이다.

향후 연구과제는 제안된 측정이 많은 사례연구를 통해 실험적 유효성 검증이 이루어져야 하며, 컴포넌트의 품질 특성별 평가 지침 마련이 필요하다.

참 고 문 헌

[1] J. S. Poulin, Measuring Software Reuse - Principles, Practices, and Economic Models, Addison-Wesley, 1997.

[2] Colin Atkinson, Joachim Bayer, Christian Bunse, Erik Kamsties, Oliver Laitenberger, Roland Laqua, Dirk Muthig, Barbara Peach, Jurgen Wust, Jorg Zettel, Component-Based Product Line Engineering with UML, Addison Wesley, 2002.

[3] George T. Heineman, William T. Councill, Component-Based Software Engineering : Putting the Pieces Together, Addison-Wesley, 2001.

[4] C. McClure, Software Reuse Techniques: A Guide to Adding Reuse to the Software Process, Extended Intelligence Inc., 1996.

[5] Clemens Szyperski, Component Software: Beyond Object-Oriented Programming, ACM Press and

Addison-Wesley, 1998.
[6] Roger S. Pressman, Software Engineering : A Practitioner' s Approach, Fifth Edition, McGraw-Hill, June 2000.
[7] 한국소프트웨어 컴포넌트 컨소시엄, 컴포넌트란 무엇인가? 알기쉬운 소프트웨어 컴포넌트, December 2001.
[8] Rational Component-Based Development Solution Seminar material, source in http://www.rational.co.kr/events/down/events/CBD_With_Rose.pdf, Auguest 2000.
[9] John Cheesman, John Daniels, UML Components: A Simple Process for Specifying Component-Based Software, Addison-Wesley, 2001.
[10] George T. Heineman, William T. Councill, Component-Based Software Engineering : Putting the Pieces Together, Addison-Wesley, 2001.
[11] Desmond F. D'Souza, Alan C. Wills, Object, Component and Framework with UML : The Catalysis Approach, Addison-Wesley, 1999.
[12] B. Henderson-Sellers, Object-oriented Metrics : Measures of Complexity, Prentice-Hall, 1996.
[13] Bindu Mehra, A Critique of Cohesion Measures in the object-Oriented Paradigm, Master Thesis, Department of Computer Science, Michigan Technological University, 1997.
[14] M. Lorenz, J. Kidd, Object-Oriented Software Metrics, Prentice Hall, 1994.
[15] Lionel Briand, Sandro Morasca, Victor Basili, "Property-based Software Engineering Measurement", IEEE Transactions on Software Engineering, Vol.22, No.1, pp.68-86, January 1996.
[16] Elaine J. Weyuker, "Evaluating software complexity measures", IEEE Transactions on Software Engineering, Vol.14, No.9, pp.1357-1365, September 1988.

고 병 선
1995년 숙명여자대학교 컴퓨터과학과 졸업(학사). 1998년 숙명여자대학교 컴퓨터과학과 이학석사. 1998년~현재 숙명여자대학교 컴퓨터과학과 박사수료. 관심분야는 품질평가, 컴포넌트 개발 방법론, 재사용

박 재 년
1966년 고려대학교 졸업(학사). 1969년 고려대학교 이학석사. 1981년 고려대학교 이학박사. 1979~1983년 전남대학교 전산통계학과 교수. 1983년~현재 숙명여자대학교 컴퓨터과학과 교수. 관심분야는 시스템 분석 및 개발 방법론, 품질평가, 시뮬레이션