Unified Process의 분석 클래스에 대한 복잡도 척도

김 유 경† · 박 재 년††

요 약

구조적 개발 방법론에 적용하도록 만들어진 복잡도 척도들은 클래스, 상속성, 메시지 전달 그리고 캡슐화와 같은 객체지향의 개념에 직접적으로 적용할 수 없다. 또한, 기존의 객체지향 소프트웨어에 대한 척도의 연구는 프로그램의 복잡도나, 설계 단계의 척도가 대부분이었다. 실제로 분석 단계 클래스의 복잡도를 낮춤으로써, 시스템의 개발 노력이나 비용 및 유지보수 단계에서의 노력이 크게 줄어들게 되므로, 분석 클래스에 대한 복잡도를 측량하기 위한 척도가 필요하다.

본 논문에서는 객체지향 개발방법론인 Unified Process의 분석 단계에서 추출되는 분석 클래스에 대하여 복잡도를 측정할 수 있는 새로운 척도를 제안한다. 협력의 복잡도 CC(Collaboration Complexity)는 가능한 협력의 최대 수로서 클래스가 잠재적으로 얼마나 복잡할 수 있는지를 측정하기 위한 척도이며, 각 협력자들의 인터페이스를 이해하는 것과 관련된 총체적 어려움을 측정하는 인터페이스 복잡도 IC(Interface Complexity)를 정의하였다. 제안된 척도는 클래스의 크기 및 상속성에 대하여 수학적인 증명을 하였으며, Weyuker의 9가지 공리적 성질에 대하여 이론적인 검증을 하였다. 또한, 텍스트 마이닝 기법을 사용하여 사용자의 질문에 자동으로 응답하는 시스템의 분석 클래스에 대하여 제안된 척도를 적용하여 복잡도를 측정하였고 기존의 복잡도 척도인 CBO와 WMC의 값을 계산하여 비교하였다. CC와 CBO, IC와 WMC의 값을 비교해 본 결과 제안된 복잡도 척도의 계산 결과 값이 그 값들보다 좀 더 복잡도를 잘 표현하고 있었다.

이로써 소프트웨어 개발 주기의 초기에 클래스에 대한 복잡도를 평가해 보고, 나머지 단계에 필요한 시간과 노력을 예측함으로써 보다 비용-효과적인 객체지향 소프트웨어를 개발할 수 있는 가능성이 높아진다.

키워드 : 객체지향, 복잡도 척도, 클래스, 메트릭

Complexity Metrics for Analysis Classes in the Unified Software Development Process

Yu-Kyung Kim† · Jai-Nyun Park††

ABSTRACT

Object-Oriented (OO) methodology to use the concept like encapsulation, inheritance, polymorphism, and message passing demands metrics that are different from structured methodology. There are many studies for OO software metrics such as program complexity or design metrics. But the metrics for the analysis class need to decrease the complexity in the analysis phase so that greatly reduce the effort and the cost of system development. In this paper, we propose new metrics to measure the complexity of analysis classes which draw out in the analysis phase based on Unified Process. By the collaboration complexity, is denoted by CC, we mean the maximum number of the collaborations can be achieved with each of the collaborator and determine the potential complexity. And the interface complexity, is denoted by IC, shows the difficulty related to understand the interface of collaborators each other. We prove mathematically that the suggested metrics satisfy OO characteristics such as class size and inheritance. And we verify it theoretically for Weyuker's nine properties. Moreover, we show the computation results for analysis classes of the system which automatically respond to questions of the it's user using the text mining technique. As we compared CC and IC to CBO and WMC, the complexity can be represented by CC and IC more than CBO and WMC. We expect to develop the cost-effective OO software by reviewing the complexity of analysis classes in the first stage of SDLC (Software Development Life Cycle).

Key word : Object-oriented, Complexity measurement, Class, Metrics

1. 서 론

측정(Measurements)은 소프트웨어 개발 주기의 각 단계에서 산출물(product)과 개발 절차(process)를 제어하고 평가하기 위하여 필요하다[1]. 또한, 소프트웨어 측정은 개발 비용이나 오류 발생률 등에 대한 예측에 사용된다. 따라서, 가능한 한 소프트웨어 개발 절차의 초기에 계산 가능한 척도를 사용하여 측정을 하게된다면, 보다 비용-효과적인 소프트웨어 개발이 이루어질 것이다. 그러나, 소프트웨어 측

※ 본 논문은 KISTEP 과제 지원에 의해 연구되었음.
† 정 회 원 : 숙명여자대학교 대학원 컴퓨터과학과
†† 정 회 원 : 숙명여자대학교 컴퓨터과학과 교수
논문접수 : 2000년 9월 21일, 심사완료 : 2001년 2월 22일

정이 효과적으로 이루어지지 못할 경우, 측정에 따르는 비용과 시간을 소비하게 된다. 이러한 소프트웨어 측정의 실패에 대한 주요 원인은 소프트웨어 개발 방법이 빠르게 변화하고 있음에도 불구하고, 측정 모델은 여전히 개선되지 않고 있기 때문이다[2]. 또한, 기존의 소프트웨어 측정에 대한 연구의 대부분이 모델에 대한 가설과 제한사항을 충분히 설명하지 못하고 있다. 본 논문에서는 이러한 문제점을 극복하기 위해 가능한 정확한 가설을 설명하고자 하였다.

Unified Process(UP)는 객체지향 개발 방법론으로서 캡슐화와 추상화, 상속성, 다형성과 같은 개념을 이용하기 때문에 기존의 절차적 방법론과는 다른 척도가 필요하다. 따라서, 본 논문에서는 UP의 분석 단계에서 추출할 수 있는 정보만을 사용하여, 클래스의 복잡도를 측정할 수 있는 복잡도 척도로서 협력의 복잡도와 인터페이스 복잡도를 제안한다. 협력의 복잡도는 클래스가 잠재적으로 얼마나 복잡할 수 있는지를 측정하기 위한 것으로서 클래스가 가지는 책임의 개수를 조사하여 척도를 제안하였다. 인터페이스 복잡도는 클래스와 협력 관계에 있는 다른 클래스들의 인터페이스를 조사하여 정의된다.

본 논문의 구성은 다음과 같다. 2장에서는 기존의 객체지향 소프트웨어에 대한 척도와 관련된 연구에 대하여 살펴보고, 3장에서는 본 논문에서 새롭게 제안하고 있는 복잡도 척도에 대하여 설명한다. 4장에서는 제안된 척도에 대한 수학적인 증명과 함께 이론적인 평가가 이루어진다. 5장은 사례 연구의 결과를 분석하고, 마지막으로 6장에서 결론 및 향후 연구과제를 기술한다.

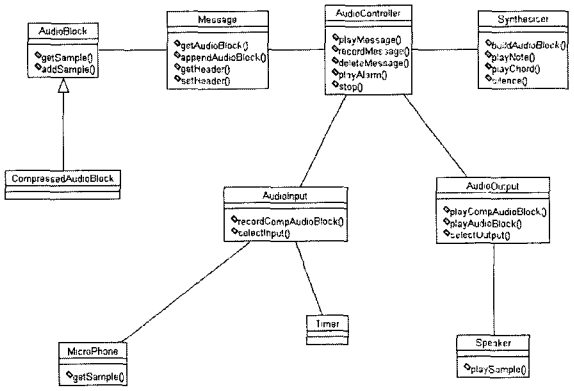
2. 관련 연구

2.1 UP의 분석 클래스

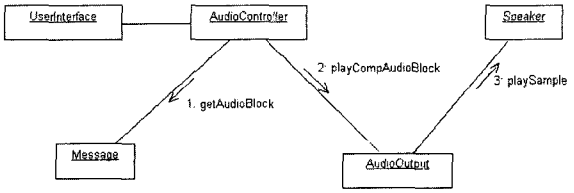
UP는 Booch 방법론과 Rumbaugh의 OMT, Jacobson의 OOSE 등의 3가지 방법론을 중심으로 여러 객체지향 방법론들이 하나로 통합되어, OMG(Object Manager Group)에서 채택된 표준 객체지향 방법론이다[3]. 이것은 사용사례 중심(Use-case Driven)이며 구조 중심(Architecture Centric)의 개발 절차이고, 또한 반복적이면서 점진적인 소프트웨어 개발 절차이다[4].

특히, 분석은 사용사례 모델을 입력으로 하여 요구사항을 구조화시키는 단계로서, 먼저 분석 패키지들 사이의 공통성 처리와 서비스 패키지를 식별하는 절차로 시작한다. 그리고 나서, 분석 패키지의 의존성을 정의하며, 개체 클래스를 식별하는 아키텍처 분석 절차로 진행된다. UP의 분석단계에서 작성되는 분석 모델은 클래스와 클래스 사이의 관계를 표현하며, 시스템의 정적인 관점을 나타낸다[4]. 분석 클래스는 설계 단계의 하나 또는 여러 개의 클래스들 또는 서브시스템의 추상화를 나타낸다. 분석 클래스는 분석 절차를 통하여 책임과 속성, 일반화 관계 및 연관관계와 구성관계를 갖도록 구성된다. 분석 과정에서 클래스도와 협력도가 작성이 되며, 이 두 다이어그램에 나타난 분석 클래스와 클래스 사이의 관계를 기반으로 복잡도 척도를 정의하였다. 본 논문에서 제안하고 있는 복잡도 척도를 분석 클래스에 적용함으로써 설계와 구현단계에서의 복잡도와 개발 노력을 예측할 수 있다.

(그림 1)과 (그림 2)는 클래스도와 협력도의 예를 보여주고 있다. UP에서는 표준 모델링 언어인 UML을 사용하여 각 다이어그램을 작성한다. (그림 1)은 Digital Sound Recorder의 클래스도의 일부이며, (그림 2)는 메시지 동작에 대한 협력도이다[5].



(그림 1) 클래스도의 예



(그림 2) 협력도의 예

2.2 객체지향 척도

클래스의 복잡도와 관련된 객체지향 척도들 가운데 가장 많이 사용되고 있는 것은 Chidamber와 Kemerer가 제안한 객체지향 설계를 위한 척도이다. 이들은 [6]에서 설계 규모와 복잡도에 영향을 주는 요소를 파악하기 위한 6가지 척도를 제시하였으며, 클래스의 복잡도에 관련된 척도로는 WMC(Weighted Methods per Class)와 CBO(Coupling Between Objects)가 있다. WMC는 클래스 당 가중치를 갖는 메소드의 수로서, 만약 모든 메소드 복잡도가 일정한 단위값을 갖게 된다면 WMC는 메소드의 개수인 n과 같다. CBO는 한 클래스와 연관되어 있는 다른 클래스의 개수로 정의된다.

[7]에서는 객체지향 패러다임의 기본 컴포넌트를 정의하고, 각각의 기본 컴포넌트가 포함하고 있는 프로그래밍 습

성(behavior)을 설명하였다. 또한, 이에 따른 5가지 척도를 제안하였다. 이 가운데 클래스의 복잡도에 관한 척도는 DAC(Data Abstraction Coupling), NOM (Number Of local Methods), 그리고 Size2이다. DAC는 클래스 내에 정의된 ADT(abstract data type)의 수로서, 클래스 내에 선언된 서로 다른 타입의 속성의 수로서 표현된다. NOM은 인터페이스의 복잡성을 표현하는 것으로서, 클래스에 의해 지역적으로 제공되는 메소드들의 수를 나타낸다. Size2는 인터페이스에서 제공되는 서비스들의 수를 나타내는 것으로서, 속성의 수와 지역 메소드의 수를 합한 것으로 정의된다.

Cohen은 [8]에서 객체지향 방법으로서 책임 기반(responsibility-driven)과 데이터 기반(data-driven)을 비교하여, [6]에서 제안한 척도들을 확장시키고 있다. 3가지 척도 가운데 클래스의 복잡도에 관한 척도는 WAC(Weighted Attributes per Class)와 NOT(Number Of Tramps)이다. WAC는 WMC를 수정 보완하여 제안된 것으로서, 클래스 안에 정의된 속성을 가중치로 구한다. 속성의 가중치는 그 크기로 정의된다. NOT는 메소드에 포함된 매개변수의 타입과 수를 포함하는 것으로서, 클래스 내에 정의된 메소드의 표현에서 외부 매개변수의 전체 개수로서 정의된다.

2.3 기존 척도들의 문제점

객체지향 분석 및 설계 단계에 적용하도록 제안된 기존의 척도들은 실제로 개발 초기 단계에서 측정할 수 있는 척도들이 거의 없다. WMC나 LCOM을 비롯하여 메소드의 매개변수나 메소드의 복잡도를 속성으로 정의된 척도들은 분석 초기에 제공되는 클래스 정보보다 훨씬 더 상세한 정보를 필요로 하기 때문에 분석 단계에서는 적용할 수 없다[9]. 객체지향 분석은 실세계의 시스템을 소프트웨어로 모델링 하여 설명하기 위한 단계로서, 문제 영역의 개념을 객체지향 모델 영역의 개념인 클래스로 추상화시키는 과정이다[10]. 즉, 문제 영역에 존재하는 중요한 대상은 클래스로 모델링 되고 시스템은 이들 클래스로부터 생성된 객체들에 의해 구축된다. 따라서, 클래스가 실세계의 대상을 부적절하게 모델링하고 있다면 시스템의 개발에 큰 장애 요소로 작용할 수 있다. 그러므로, 개발 생명주기의 가장 초기 단계에서 개발자들이 적절하게 클래스로 모델링을 하고, 올바른 클래스 분할을 선택할 수 있도록 방향을 제시하는 것은 매우 중요한 일이다. 그러므로, 분석 단계에서 개발자들이 쉽게 적용할 수 있는 척도가 요구된다.

또한, 기존의 척도들이 단순한 객체지향의 개념과 척도를 정의하는데 그치고 있어서 척도를 적용하기 위한 구체적인 절차와 가설에 대한 충분한 설명이 없다는 문제점을 가지고 있다. 이들은 척도를 정의하는데 있어서 수학적 명세를 사용하지 않아서 이해하기 어려우며, 계산하는 자세한 절차를 설명하지 않으므로 적용하는데 혼란을 야기할 수 있다[11]. 따라서, 모델에 대한 가설과 제한사항을 충분히 설명하여 개발자들이 이해하기 쉽고 적용 방법을 혼돈하지 않도록 해야할 것이다.

본 논문에서 제안하고 있는 척도의 구별되는 특징은 분석 초기의 정보인 클래스의 책임과 협력 관계에 초점을 맞추었다는 점이다. 또한, 기존의 척도들이 가지고 있는 문제점을 해결하기 위하여 이해하기 쉬운 수학적 명세를 사용하였고, 척도를 정의하기 위해 필요한 가설을 설명하고 있다.

3. 클래스 복잡도 척도

3.1 클래스 척도에 대한 가설

주어진 문제 영역(domain)은 식별가능하고 상호 작용을 하는 물리적 또는 추상적 개념인 클래스로 구성되어 있다. 객체지향 개발 단계의 초기에 문제 영역을 표현하는 클래스들은 소프트웨어 시스템의 구현된 클래스로 대응된다. 각각의 클래스는 자신이 제공하는 일련의 서비스들에 의하여 설명된다. 이러한 일련의 서비스들을 책임(Responsibilities)이라고 부른다. 책임을 수행하기 위하여 한 클래스가 다른 클래스와 상호 작용할 경우, 이러한 상호작용을 협력(Collaborations)이라고 하며 협력하게 되는 그 클래스들을 협력자(Collaborators)라고 정의한다. 분석 클래스들 사이의 상호작용은 연관 관계(Association)와 집합 연관 관계를 표현하는 구성 관계(Aggregation)로 나타나며, 이들을 협력으로 정의한다.

또한, 시스템 내의 클래스들 사이에는 상속 계층구조가 형성된다. 클래스는 부모 클래스로부터 책임을 상속받게 되며, 이러한 책임을 상속받은 책임(Inherited responsibilities)이라고 정의한다. 상속받은 책임은 자녀 클래스의 인터페이스의 일부분이 된다. 재 정의된 책임은 상속받지 않은 것으로 간주하며, 책임의 나머지 부분은 상속받은 책임을 제외한 클래스 내에서 정의된 책임들이다. 본 논문에서는 한 클래스의 책임이란 상속받은 책임을 제외한 것으로 클래스 내에서 정의된 책임들을 표현하는 항목으로 사용할 것이다. 또한, 전체 책임(Total responsibilities)이란 상속받은 책임과 상속받은 책임을 제외한 클래스의 책임을 모두 합하여 표현하는 것으로 사용한다.

제안된 척도는 한 시스템 내에서 서로 연결된 클래스들 사이의 협력을 기반으로 하고 있다. 클래스에 대한 복잡도는 개별적인 클래스 각각에 대한 복잡도뿐만 아니라 다른 클래스들과의 상호 연결성 사이에 함수관계가 존재하기 때문이다.

다음은 분석 클래스에 대한 복잡도 척도를 정의하기 위한 가설로서, 클래스의 책임 및 협력과 복잡도 사이의 관계를 설명하기 위한 것이다.

가설 1. 상속받은 책임을 제외한 클래스 책임의 개수와 협

력자의 수가 많은 클래스는 그 수가 적은 클래스보다 개발하기가 좀 더 복잡하다.

증명 : 클래스의 개발자들은 열거된 책임을 구현해야 하므로, 복잡도는 책임의 목록이 많아질수록 증가하게 된다. 또한 각 협력자와 상호작용하기 위하여, 클래스의 개발자들은 협력자의 인터페이스를 이해해야하는 부가적인 노력을 들여야한다. 그러므로, 개발의 복잡도는 협력자의 수가 많아질수록 증가하게 된다. 따라서, 책임의 개수 및 협력자의 개수와 개발 복잡도 사이에는 다음 식 (1)과 같은 관계가 성립한다.

$$C(E) = a \cdot x_1 + b \cdot x_2 \quad (1)$$

여기에서, $C(E)$는 클래스 E를 개발할 때의 복잡도이고, a, b는 상수로서 $a>0, b>0$이다. x_1은 클래스의 책임의 개수이고, x_2는 협력자의 개수이다.

임의의 한 클래스 E의 상속받은 책임을 제외한 책임의 개수와 협력 관계에 있는 협력자의 개수를 각각 r_E, c_E라고 하자. 그렇다면, 두 클래스 F, G의 복잡도는 식 (1)로부터 각각 $C(F) = a \cdot r_F + b \cdot c_F$와 $C(G) = a \cdot r_G + b \cdot c_G$가 된다. 만약, 클래스 F의 책임의 개수와 협력자의 개수가 더 많다면, $r_F > r_G, c_F > c_G$가 성립한다. 따라서, $a>0$이고 $b>0$이므로, $C(F) = a \cdot r_F + b \cdot c_F > a \cdot r_G + b \cdot c_G = C(G)$가 성립한다.

그러므로, 상속받은 책임을 제외한 클래스 책임의 개수와 협력자의 개수가 많은 클래스는 개발하기가 좀 더 어려워진다는 것을 증명할 수 있다.

가설 2. 책임의 전체 개수가 많은 클래스는 그 수가 적은 클래스보다 사용하기가 좀 더 복잡하다.

증명 : 한 클래스를 사용하기 위해서는 그 클래스의 인터페이스를 이해해야만 한다. 클래스의 인터페이스는 그 클래스의 책임뿐만 아니라, 상속받은 책임을 포함한다. 또한, 협력자의 인터페이스도 이해해야 그 클래스를 사용할 수 있게 된다. 따라서, 사용상의 복잡도는 다음 식 (2)와 같이 정의할 수 있다.

$$U(E) = a \cdot (x_1 + x_2) \quad (2)$$

여기에서, $U(E)$는 클래스 E의 사용상의 복잡도이며, a는 상수로서 $a>0$이다. x_1은 클래스 E의 전체 책임의 개수이고, x_2은 협력자들이 갖는 전체 책임의 총 개수이다.

임의의 한 클래스 E의 전체 책임의 개수를 R_E라고 하자. 그렇다면, 두 클래스 F, G의 복잡도는 식 (2)로부터 각각 $U(F) = a \cdot R_F + R_{c_F}$와 $U(G) = a \cdot R_G + R_{c_G}$가 된다. 만약, 클래스 F의 전체 책임의 개수가 더 많다면, $R_F > R_G$이고, $R_{c_F} \geq R_{c_G}$가 성립한다. 따라서, $a>0$이므로 $U(F) = a \cdot R_F + R_{c_F} \geq a \cdot R_G + R_{c_G} = U(G)$이다.

그러므로, 클래스 전체 책임의 개수 많은 클래스는 사용하기가 좀 더 어려워진다는 것을 증명할 수 있다.

가설 3. 개발 복잡도는 상속받은 책임을 제외한 클래스 책임의 개수와 협력자의 수가 증가하는 것에 대하여 비 선형적(non-linearly)으로 증가한다.

증명 : 식 (1)은 종속 변수인 개발 복잡도와 독립 변수인 책임의 개수 및 협력자의 개수 사이의 함수 관계를 표현하고 있다. 이 식 (1)에서 볼 수 있듯이, 이 변수들 사이에는 비 선형적인 관계가 존재함을 알 수 있다.

가설 4. 사용상(usage)의 복잡도는 책임의 전체 개수가 증가하는 것에 대하여 비 선형적으로 증가한다.

증명 : 식 (2)에서는 종속 변수인 사용 복잡도와 독립 변수인 전체 책임의 개수 사이에 존재하는 함수 관계를 표현하고 있다. 여기에서 전체 책임의 개수가 증가하면 사용의 복잡도가 증가하지만, 선형적으로 증가하지는 않는다는 것을 알 수 있다.

가설 5. 한 클래스는 항상 자기 자신과 협력할 수 있다.

증명 : 이것은 클래스 내에 있는 어떤 책임도 자기 자신 또는 다른 어떤 책임을 재귀적으로 호출할 수 있다는 사실 때문에 매우 명백해진다.

3.2 협동 그래프 COG(COllaboration Graph)

클래스들 사이의 협력 관계와 상속 관계를 표현하고, 복잡도 척도를 계산하기 위하여, 분석 클래스의 정보를 표현하고 있는 클래스도와 교류도인 협력도로부터 협동 그래프 COG를 다음과 같이 정의한다.

정의 1-1. COG는 유방향 그래프(Directed Graph)로서, COG = (V, E)이다. 여기에서, V는 정점의 유한 집합이고, E는 간선의 유한 집합이다. 또한, $V \neq \emptyset$이며, $E \subseteq V \times V$이다.

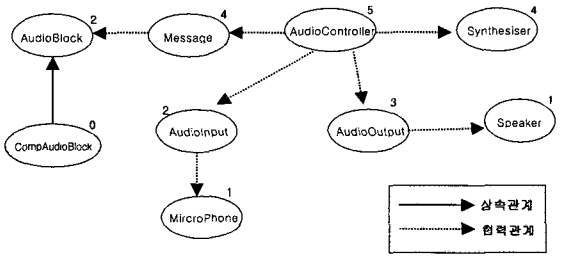
정의 1-2. 임의의 한 분석 모델 M을 위한 협동 그래프는 COG = (V, E)이며, 여기에서 V는 M에 나타난 클래스를 표현하는 정점들의 집합이다. 각 정점은 가능한 책임의 수를 표현하는 레이블을 가지고 있으며, 그 값은 정수(integer)이다.

정의 1-3. 임의의 한 분석 모델 M을 위한 협동 그래프는 COG = (V, E)이며, E는 M에 나타난 클래스들의 관계를 표현하는 간선들의 집합으로, $E = E_c \cup E_i$이고, E_c는 협력 관계, E_i는 상속 관계를 나타낸다.

협동 그래프 COG는 UP의 분석 모델인 클래스도와 협력도로부터 생성될 수 있으며, CRC(Class-Responsibilities-Collaboration)기법과 같은 객체지향 분석 과정의 또 다른

산출물에도 적용할 수 있다. 클래스도의 클래스는 COG의 정점으로 나타나게 되며, 클래스의 이름을 같이 기술하게 된다. 각 정점은 상속받은 책임을 제외한 클래스 책임의 개수를 나타내는 정수 레이블을 갖는다. 또한, 클래스도에 나타난 상속 관계는 COG에서 부모 클래스로 향하는 점선으로 표현이 된다. 클래스 사이의 협력 관계를 나타내는 연관 관계와 구성 관계는 클래스도와 협력도를 참조하여 COG에 표현하게 되는데, 메시지의 전달이 이루어지는 클래스에 대하여 방향을 갖는 실선으로 나타낸다.

앞의 (그림 1)과 (그림 2)로부터 생성되어지는 COG는 (그림 3)과 같다. 아홉 개의 클래스는 정수의 레이블을 갖는 정점으로 표현되어 원형으로 표현이 되며, 원의 내부에 클래스의 이름을 기술한다. 상속 관계는 부모 클래스인 클래스로 향하는 점선으로, 협력 관계는 협력자로 향하는 실선으로 나타난다.



(그림 3) COG의 예

COG는 클래스의 책임과 협력자에 대한 정보를 제공하며, 상속 관계와 협력 관계를 쉽게 파악하여 복잡도를 계산할 수 있게 된다.

3.3 복잡도 척도

협력의 복잡도는 가능한 협력의 최대 수로서 클래스가 잠재적으로 얼마나 복잡할 수 있는지를 알려주게 된다. 그러나 이것을 정확하게 알 수 없기 때문에, 구현되어지는 책임 각각에 대하여 협력자들이 사용된다고 가정한다. 따라서, 한 클래스 E에 대한 협력의 복잡도를 CC라고 표현한다면, 다음의 정의 2-1과 같다.

정의 2-1. $CC(E) = r\,(1+c)$

가설 3에서는 r과 c의 값에 대하여 복잡도가 비 선형적인 의존성을 가지고 있다. 척도가 비 선형적 성질을 만족하고 있다는 것을 알 수 있다. 여기에서 협력자의 수가 (1+c)가 된 것은 가설 5에 의하여 클래스는 자기자신과도 협력하기 때문이다.

인터페이스 복잡도는 각 협력자들의 인터페이스를 이해하는 것과 관련된 총체적 어려움을 표현한다. 따라서, 인터페이스 복잡도는 개별 협력자들의 인터페이스 크기를 더하여 구할 수 있다.

각 클래스에 대한 인터페이스 크기는 R 즉, 책임의 전체 개수가 된다. 그러나 이것은 인터페이스 복잡도와 R 사이의 비 선형 관계를 가정한 가설 4에 위배된다. 따라서, R이 상수 지수 m에 의해 증가되는 것으로 인터페이스 복잡도를 측정하도록 정의하였다. 계산의 편의를 위하여 m을 2로 선택하였으며, 따라서, 임의의 한 클래스의 인터페이스 크기는 R^2이 된다.

클래스 E가 전체 R개의 책임을 가지고 있고, c개의 다른 클래스들 $E_1, E_2, \cdots, E_c$과 협력하고, 인터페이스 크기를 IS라고 표현한다면, 인터페이스 복잡도 IC는 정의 2-2와 같이 주어진다.

정의 2-2. $IC(E) = R^2 + \sum_{i=1}^{c} IS(E_i)$

여기에서, $IS(E_i)$는 클래스 E_i의 인터페이스 크기를 나타낸다.

인터페이스 복잡도에서 R이 상수 지수 m에 의해 증가되는 것으로 정의하였고, 계산의 편의를 위하여 m을 2로 선택했다. 이 관계는 명확하지 않다. 따라서, m에 대한 다른 가능한 공식을 가지고, ATM 클래스[12]에 대하여 유의 수준 5%로 켄달(Kendall)의 순위 검정방법을 이용하여 시험하였다. 켄달의 순위 상관 계수 τ를 사용하여 R^2과 R^3, R^4, R^5, R^6 사이의 관계를 조사하였다. 아래 <표 1>에서 보듯이, 상관 관계에서 매우 미세한 차이를 보이고 있으며 따라서 좀더 구현이 간단해 지도록 R^2을 선택한 것에 대한 분명한 이유를 제시했다고 할 수 있다.

〈표 1〉 R에 대한 검정의 결과

R^m	τ	유의 수준
R^3	0.65	0.05
R^4	0.66	0.05
R^5	0.66	0.05
R^6	0.66	0.05

4. 복잡도 척도에 대한 증명

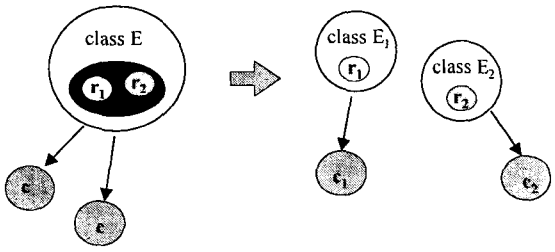
클래스 기반의 복잡도 척도에 대하여 조사되어야 할 두 가지 중요한 측면으로서 클래스의 크기와 상속성에 대하여 어떻게 반응하는 가에 관한 것이다.

4.1 클래스 크기

클래스의 크기는 결합과 분할에 대하여 생각할 수 있다. 객체지향 실무자들은 너무 커서 다루기 어려운 클래스의 사용을 피하고, 예외적으로 너무 많은 수의 메소드를 가진 클래스에 대해서는 좋지 않은 설계로 간주한다[12]. 이렇게 큰 클래스들은 좀 더 분할될 수 있다. 또한, 장기적으로 시스템의 오버헤드가 증가하기 때문에 너무 작은 클래스도

역시 좋지 않다. 따라서, 복잡도 척도는 클래스의 최적의 크기를 판단할 수 있어야 하고, 제안된 척도가 클래스 분할과 결합의 경우에 클래스 크기에 영향을 주는지를 살펴보고자 한다.

책임의 집합 P와 협력자의 집합 C를 가지는 클래스 E를 n개의 책임과 n개의 클래스를 갖도록 $P_1, P_2, \cdots, P_n$과 $E_1, E_2, \cdots, E_n$으로 나눌 때, 이것을 분할이라고 정의한다. 여기에서, $P = \bigcup_{i=1}^{n} P_i$이고, 임의의 두 부분집합 P_i, P_j에 대하여 $P_i \cap P_j = \emptyset$이다. 또한, $C = \bigcup_{i=1}^{n} C_i$이고, $C_i \cap C_j = \emptyset$이다.



(그림 4) 클래스의 분할

위의 (그림 4)에서, 클래스 E는 r_1과 r_2로 분할되는 r개의 책임과 c개의 협력자를 가지고 있으며, 어떤 다른 클래스로부터 상속받지 않으므로, $r = R$이다. 이 클래스 E가 각각 r_1과 r_2의 책임을 갖고, c_1과 c_2의 협력자를 갖는 좀 더 작은 클래스들 E_1과 E_2로 분할되었다. 따라서, $r = r_1 + r_2$이고, $c = c_1 + c_2$가 된다는 것을 알 수 있다. 이러한 경우, 클래스 E에 대한 CC를 계산하고 분할된 경우의 CC와 비교하여 보자.

$$\begin{aligned} CC(E) &= (c+1)r = (c_1 + c_2 + 1)(r_1 + r_2) \\ &= (c_1 + 1)r_1 + (c_2 + 1)r_2 + c_2 r_1 + c_1 r_2 \\ &> CC(E_1) + CC(E_2) \end{aligned}$$

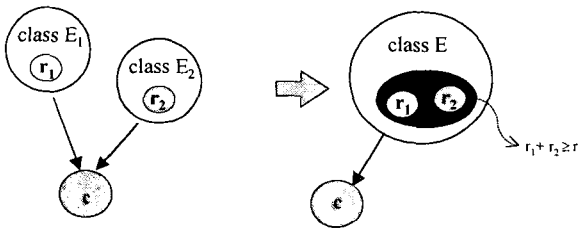
위의 식에서 알 수 있듯이, CC의 경우 클래스가 두 클래스로 분할 될 경우 좀더 낮은 복잡도를 갖게 된다.

또한, (그림 4)와 같이 E의 협력자의 집합 $\{F_1, \cdots, F_c\}$가 집합$\{F_1, \cdots, F_{c_1}\}$과 $\{F_{c_1+1}, \cdots, F_c\}$인 각각 두 클래스 E_1과 E_2의 협력자의 집합으로 분할되었다면, 다음과 같이 IC를 계산할 수 있다. 역시 클래스가 분할될 경우 좀더 낮은 복잡도를 갖게 된다는 것을 알 수 있다.

$$\begin{aligned} IC(E) &= r^2 + \sum_{i=1}^{c} IC(F_i) \\ &= (r_1 + r_2)^2 + \sum_{i=1}^{c_1} IC(F_i) + \sum_{k=c_1+1}^{c} IC(F_k) \\ &= {r_1}^2 + \sum_{i=1}^{c_1} IC(F_i) + {r_2}^2 + \sum_{k=c_1+1}^{c} IC(F_k) + (2r_1 r_2) \\ &= IC(E_1) + IC(E_2) + 2r_1 r_2 \\ &> IC(E_1) + IC(E_2) \end{aligned}$$

다음은 클래스 결합의 경우, 같은 협력자 집합을 가지고 있는 공통성이 많은 두 클래스들을 한 클래스로 결합할 때, 복잡도에 어떤 영향을 주게 되는지를 보이고자 한다.

(그림 5)에서 두 클래스 E_1과 E_2는 각각 r_1과 r_2의 책임을 가지며, $c = |C|$인 공통 협력자의 집합 $C = \{F_1, F_2, \cdots, F_c\}$를 가지고 있다. 인터페이스의 복잡도 $\sum_{i=1}^{c} IC(F_i)$는 이 예제의 모든 클래스들에 대해 일정한 상수 값을 갖게 되므로, δ로 표시한다.



(그림 5) 두 클래스의 결합

두 클래스를 하나의 새로운 클래스 E로 결합할 경우, E의 책임의 집합은 E_1과 E_2의 책임을 결합한 것이다. 그러므로, 클래스 E가 갖는 책임의 개수는 공통의 책임이 없다면 $r = r_1 + r_2$이 되며, 그렇지 않은 경우 $r < r_1 + r_2$가 된다. 원래의 경우와 결합한 경우의 상대적인 복잡도를 계산하기 위하여 CC를 고려해 보자. 세 클래스 E, E_1 그리고 E_2에 대하여 CC는 다음과 같은 관계가 있다.

$$\begin{aligned} CC(E) &= r(c+1) \le (r_1 + r_2)(c+1) \\ &= r_1(c+1) + r_2(c+1) = CC(E_1) + CC(E_2) \end{aligned}$$

결과에서 볼 수 있듯이, 두 클래스가 결합된 경우 CC의 복잡도가 줄어들게 된다.

클래스의 결합에 대한 IC의 값을 비교하여 보자. (그림 5)와 같은 클래스 결합의 경우를 생각할 때 IC는 다음과 같아진다.

$$\begin{aligned} IC(E_1) + IC(E_2) &= {r_1}^2 + {r_2}^2 + 2\delta \\ &\ge r^2 - 2r_1 r_2 + 2\delta \\ IC(E) &= r^2 + \delta \end{aligned}$$

만약, $\delta > 2r_1 r_2$라면 $IC(E) < IC(E_1) + IC(E_2)$이므로 두 클래스가 결합된 경우의 복잡도가 더 작다는 것을 알 수 있다. 그렇지 않다면, 결합된 경우의 복잡도는 크거나 같아지게 된다.

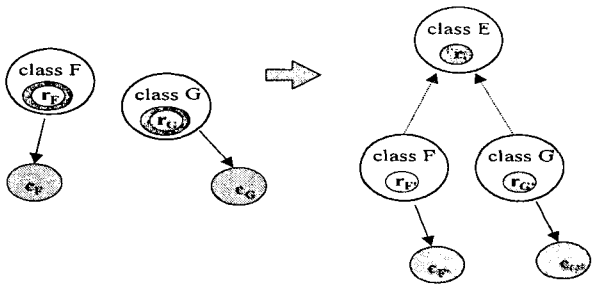
클래스 E_1과 E_2가 각각 c개의 협력자들을 가지고 협력

하고 있다면, 같은 협력자의 집합을 갖는 협력의 복잡도는 두 가지 서로 다른 장소에서 발생하는 것이다. 클래스의 결합은 δ인자에 의하여 복잡도를 감소시키게 되지만, 클래스 E에 있는 책임들을 결합함으로써, $2r_1r_2$ 인자에 의하여 복잡도의 증가를 가져오게 된다. 따라서, 어느 쪽이 더 커지느냐에 따라서 척도는 결합을 허락할 수도 혹은 인정하지 않을 수도 있다. 그러나, 두 경우 가운데 어떤 것이 더 좋은지를 추측하기 전에 전체 구조에 대한 관점을 가지고 결합되지 않은 클래스와 결합된 클래스들을 평가할 필요가 있다.

4.2 상속성

객체지향 시스템에서, 상속성은 서브타입의 관계를 모델링 하기 위하여 주요하게 사용된다. 부모 클래스의 모든 책임은 자녀 클래스들이 상속받을 수 있으며, 이렇게 상속되는 책임들은 자녀 클래스들 내에서 재정의 될 수 있다. 상속 메커니즘은 객체지향 시스템을 변화에 민감하지 않고, 따라서 좀더 유지보수하기 쉽게 만드는 것으로서 복잡도를 감소시키는데 매우 중요한 것이다.

제안된 척도에 대한 상속성의 영향을 시험해보기 위하여 상속받기 전과 후의 값을 비교해 보았다. (그림 6)에 보이는 것처럼, 두 클래스 F와 G는 각각 책임 r_F와 r_G, 그리고 협력자 집합 c_F와 c_G를 가지고 있다. 이들 클래스의 공통의 책임을 r이라고 하면, 그림의 오른쪽과 같이 부모 클래스 E로부터 상속을 받고 있는 상속 계층구조를 이룰 수 있다.



(그림 6) 상속 전후의 두 클래스

클래스 E는 클래스 F와 G에 정의된 책임들의 일부로 구성된 일련의 r개 책임들을 수행하게 된다. 그러면, $r_F = r_{F'} + r$이고, $r_G = r_{G'} + r$이 된다. $|c_F|$와 $|c_G|$를 각각 F와 G의 협력자들의 개수가 되고, $|c_F| = |c_F'|$이고, $|c_G| = |c_G'|$이다. 이와 같은 경우 CC는 다음과 같다.

$$CC(F) = r_F(1+|c_F|) > r_{F'}(1+|c_F|) = r_{F'}(1+|c_F'|) = CC(F')$$
$$CC(G) = r_G(1+|c_G|) r_{G'}(1+|c_G|) = r_{G'}(1+|c_G'|) = CC(G')$$

따라서, 상속을 받도록 구성된 경우의 자녀 클래스는 협력의 복잡도가 감소하게 된다는 것을 알 수 있다.

인터페이스 복잡도 IC가 상속성에 어떻게 반응하는지를 살펴보기 위하여, δ_F와 δ_G를 클래스 F와 G의 협력자 집합에 대한 IC의 합이라고 하자. 그리고, (그림 6)의 클래스 F와 클래스 F′의 인터페이스 복잡도를 비교하면 다음과 같다.

$$IC(F) = r_F^2 + \delta_F = (r_{F'} + r)^2 + \delta_F = IC(F')$$

그러므로, 상속된 경우의 인터페이스 복잡도는 같아진다는 것을 알 수 있다. 이것은 인터페이스 복잡도가 클래스가 갖는 전체 책임의 개수에 기반을 두고 정의되었기 때문이다. 상속이 이루어진다고 해도 전체 책임의 개수는 변하지 않기 때문에 복잡도에 변화가 없는 것이다.

4.3 Weyuker의 성질을 이용한 분석적 평가

Weyuker[13]는 절차적(procedural) 프로그램을 위한 복잡도 척도에 요구되는 성질들을 정의하였다. 이 성질들은 Cherniavsky[14] 등에 의하여 비판을 받고 있지만, 엄격한 이론적 검증방법을 제공하고 있다[15]. Weyuker에 의해 보여진 것처럼, 이 성질들을 이용하여 본 논문에서 제안하고 있는 복잡도 척도를 검증할 수 있을 것이다.

Weyuker의 성질에서 사용되는 E ; F는 서로 다른 임의의 클래스 E와 F의 결합을 나타내며, $|E|$는 클래스 E의 복잡도로서 $|E| \geq 0$이다.

Property 1. $(\exists E)(\exists F)\ (|E| \neq |F|)$.

이 성질은 모든 클래스들이 같은 척도 값을 가지 수 없음을 나타내고 있다. 다시 말하자면, 서로 다른 복잡도를 갖는 클래스들이 적어도 두 개는 존재한다는 것을 의미한다.

제안된 척도는 클래스에 정의된 책임의 개수와 상속 및 협력 관계에 있는 클래스의 개수에 기반을 두고 정의되었다. 일반적으로, 서로 다른 두 개의 클래스가 같은 협력 클래스들을 갖으며, 같은 상속 관계를 유지하는 경우는 매우 드문 일이다. 따라서, 대부분의 서로 다른 두 클래스는 서로 다른 CC 값을 가지게 되며, 이로써 $E \neq F$에 대하여 $CC(E) \neq CC(F)$이라는 사실을 알 수 있다. 또한 IC의경우, 책임의 개수에 대한 제곱으로 계산을 하게 된다. 서로 다른 클래스 E와 F에 선언된 책임의 수는 서로 독립적이며, 동등하게 분산되어 있다. 이것은 $|E| \neq |F|$인 어떤 임의의 클래스 F가 존재할 수 있다는 가능성을 의미한다. 따라서, 대부분의 서로 다른 두 클래스는 서로 다른 IC 값을 가지게 되며, 이로써 성질 1을 만족한다는 것을 알 수 있다.

Property 2. c를 음이 아닌 수라고 하면, 주어진 문제 영역 내에서 복잡도가 c인 클래스는 단지 유한하게 존재한다.

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

한다는 것이다. [6]에서는 일반적으로 모든 시스템이 유한개의 클래스로 이루어지므로 클래스 수준에서 측정된 어떤 척도에 의해서도 충족될 것이라고 설명하고 있다. 제안된 척도의 경우, CC와 IC의 값이 상수로 주어진다면 CC는 클래스에 대한 c와 r값의 조합만큼 유한한 수가 존재하게 된다. 따라서, 성질 2를 만족하게 된다. IC는 클래스에 대한 R값과 그 협력자에 대한 R값의 조합만큼 유한한 수가 존재하게 된다. 따라서, 성질 2를 만족하게 된다.

Property 3. $(\exists E)(\exists F)\ (|E|=|F|)$.

이것은 같은 복잡도를 갖는 서로 다른 두 클래스가 존재한다는 것을 의미한다. 즉, 두 클래스가 같은 설계 특성을 가질 수 있다는 것이다. 제안된 척도 정의는 수식으로 표현되기 때문에, 같은 척도 값을 갖는 클래스를 찾을 수 있다. 주어진 한 클래스 E가 가지는 책임들 가운데 한 책임의 내용(text)을 바꾼다면, 같은 척도 값을 갖는 새로운 클래스 F를 얻을 수 있다. 그러므로 제안된 척도 CC와 IC는 이 성질을 만족한다.

Property 4. $(\exists E)(\exists F)\ (E\equiv F \text{ and } |E|\neq|F|)$.

이 성질은 서로 같은 기능을 수행하지만, 복잡도가 같지 않은 두 클래스가 존재한다는 것을 나타낸다. 정확하게 서로 같은 인터페이스를 제공하는 동등한 두 클래스들이 서로 다른 내부 메커니즘을 가질 수 있으므로 제안된 척도에 대하여 이 성질은 성립한다. 다시 말하면, 같은 인터페이스를 제공한다고 해도 서로 다른 상속 계층이나 서로 다른 협력자의 집합을 가질 수 있다는 것이다. 따라서, 협력의 복잡도나 인터페이스의 복잡도는 달라질 수 있으므로 성질 4는 성립하게 된다.

Property 5. $(\forall E)(\forall F)\ (|E|\leq|E\ ;\ F| \text{ and } |F|\leq|E;F|)$.

이것은 두 클래스의 결합에 대한 복잡도가 두 요소 클래스의 복잡도보다 결코 작아질 수 없다는 것을 의미한다.

두 클래스 E와 F 각각의 협력의 복잡도 CC를 $|E|=r_E(c_E+1)$이고, $|F|=r_F(c_F+1)$라고 하자. 그리고 이 두 클래스가 결합된 경우를 살펴보면, $|E;F|=|E|+|F|-\partial$이 된다. 여기에서 ∂은 결합으로 인해 감소되는 부분을 말하며, 이로써 $|E|-\partial\geq 0$이고 $|F|-\partial\geq 0$임은 명백해진다. 따라서, $|E|+|F|-\partial\geq|E|$이고, $|E|+|F|-\partial\geq|E|$이 성립한다. 즉, 모든 클래스 E와 F에 대하여 $|E;F|\geq|E|$이고, $|E;F|\geq|F|$가 된다. 그러므로 CC에 대하여 성질 5는 성립한다. IC의 경우, 두 클래스 E, F가 갖는 책임의 개수가 $|E|=n_E$이고, $|F|=n_F$라고 하자. 그러면, 이 두 클래스가 결합된 경우 $|E;F|=n_E+n_F-\partial$이 된다. 이때, ∂은 두 클래스 E와 F의 공통으로 갖는 책임의 개수이며, 최대 값은 $\min(n_E, n_F)$가 된다.

따라서, $|E;F|\geq n_E+n_F-\min(n_E, n_F)$이므로 $|E;F|\geq|E|$이고 $|E;F|\geq|F|$임이 명백해진다. 이로써, IC에 대해서도 성질 5가 성립한다는 것을 알 수 있다.

Property 6. $(\exists E)(\exists F)(\exists G)(|E|=|F| \text{ and } |E;G|\neq|F;G|)$.

기본적으로 성질 6은 같은 복잡도를 갖는 두 클래스 E, F가 다른 클래스 G와 각각 결합되었을 경우, 결합의 순서와 상관없이 결합된 클래스는 서로 다른 복잡도를 갖게 된다는 것을 의미하고 있다. 이것은 비록 E와 F가 같은 복잡도를 갖는다고 해도, 협력자의 수나 책임의 수 또는 상속 계층 구조와 같은 그들의 개별적 속성은 다르기 때문에, 클래스 G와 결합이 이루어진 후의 $|E;G|$와 $|F;G|$는 매우 달라지게 된다는 것을 말한다.

CC의 경우 $|E|=|F|=n$인 두 클래스가 $|G|=m$인 클래스와 결합한다면, $|E;G|=n+m-\partial$이고, $|F;G|=n+m-\phi$가 된다. 여기에서 ∂과 ϕ는 각각 결합으로 인하여 감소되는 부분이며, 서로 다른 기능을 수행하므로, 같지 않다는 것을 알 수 있다. 따라서, $|E;G|\neq|F;G|$임이 성립한다. IC의 경우에도 책임의 개수가 같은 $|E|=|F|=n$인 두 클래스에 대하여 클래스 E와 공통의 책임 ∂개를 가지고, F와 공통의 책임 ϕ개를 갖는 $|G|=m$인 클래스와 결합한다고 하고, 여기에서 $\partial\neq\phi$이다. 그러면, $|E;G|=n+m-\partial$이고, $|F;G|=n+m-\phi$가 된다.

따라서, $|E;G|\neq|F;G|$임이 성립하며 성질 6을 만족하게 됨을 알 수 있다.

Property 7. E의 책임이나 메소드의 순서 배치에 의하여 F가 구성되고, $|E|\neq|F|$인 클래스 E와 F가 존재한다.

[6]과 [13]에서는 객체지향 설계에서 클래스들은 문제 영역을 추상화한 것이며, 따라서 클래스 정의 내에서 명령문의 순서는 실행하거나 사용하는데 아무런 영향을 주지 않는다고 설명하고 있다. 그러므로 이것은 클래스 수준에서 정의된 어떤 척도에 대해서도 성립하게 된다고 설명하고 있다. 이 성질은 제안된 척도가 클래스에 대한 책임의 수와 협력 클래스의 개수만을 고려하고, 책임의 순서에 대한 배열은 고려하지 않기 때문에 성립하게 된다.

Property 8. E가 F를 다시 명명한 것이라면, $|E|=|F|$이다.

이 성질은 클래스의 이름을 변화시킨다고 해도 클래스의 복잡도에는 영향을 주지 않아야 한다는 것을 의미한다. 제안된 척도의 측정은 클래스의 이름에 기반을 두지 않았기 때문에 이 성질을 만족하고 있다는 것은 명백하다.

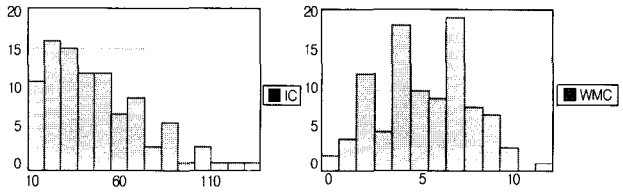
Property 9. $(\exists E)(\exists F)[(|E|+|F|)\leq(E;F)]$.

이것은 결합된 클래스가 구성 요소보다 좀 더 복잡해진다는 것을 명시하고 있다. 그러나, 앞의 복잡도에 대한 수학적 증명과정에서 클래스의 결합이 이루어질 경우, 결합된 클래스의 복잡도가 낮아진다는 것을 보았다. 따라서, 결합된 클래스의 복잡도가 증가한다는 것은 성립하지 않는다.

[6]에서는 성질 9가 객체지향 소프트웨어 설계에 대한 복잡도 척도의 기본 특성으로서는 적합하지 않으며, 성질 9를 만족하는 것이 객체지향 설계에 대한 척도로서 널리 사용되기에 오히려 더 해롭다고 설명하고 있다. 이러한 논점에서 보면, 제안된 척도는 객체지향 척도로서의 기본적인 특성을 모두 갖추었다고 할 수 있다.

5. 사례 연구 및 분석

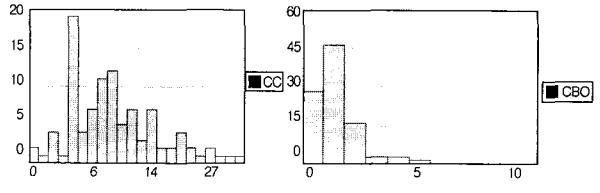
국내 S 기업과 숙명여자대학교 소프트웨어 공학 연구실 공동으로 텍스트 마이닝 기법을 이용하여 사용자의 질문에 자동으로 응답하는 시스템을 RUP(Rational Unified Process)를 적용하여 개발하였다. 본 연구에서 제안한 복잡도 척도를 적용하여 이 시스템의 분석 모델과 설계 모델을 이용하여 실험하였다. 분석 모델의 클래스의 수는 98개로서, 각 클래스에 대하여 CC와 IC를 계산하였고, 현재 가장 많이 사용되고 있는 CK척도 가운데, 클래스 복잡도를 측정하는 WMC와 CBO 값을 계산하였다. (그림 7)은 IC와 WMC의 계산된 결과 값을 비교한 그래프이다.



(그림 7) IC와 WMC 값의 분포

IC는 클래스 내에 있는 책임의 개수가 늘어남에 따라 큰 폭으로 증가하는 경향이 있다. 이것은 책임의 개수가 증가함에 따라 얼마나 더 복잡해지는지를 쉽게 판단할 수 있는 지표가 될 수 있다. WMC의 경우는 단순히 클래스가 가지고 있는 책임의 개수가 되므로, 그 값의 범위가 7±2의 범위를 크게 벗어나지 않고 있다.

CC와 CBO의 계산 결과를 비교한 그래프는 (그림 8)에 나타나 있다. CC는 외부 클래스와의 결합정도를 나타내는 것이기 때문에 CC의 값이 낮은 경우, CBO의 값도 작다는 것을 알 수 있었다. CC와 CBO의 상관관계가 높지만, 몇몇 클래스에서 다른 결과가 나왔는데, 그러한 경우는 CBO의 값은 작지만 클래스의 책임이 많은 경우이다.



(그림 8) CC와 CBO 값의 분포

그러나, CC의 값이 고르게 분포되어 있는 반면, CBO의 값은 2보다 큰 값은 거의 없이 분포되어 있음을 알 수 있다. 이렇게 값이 차이가 나는 것은 CBO가 단순히 참조하고 있는 외부 클래스의 개수만으로 복잡도를 평가하였기 때문이며, CC는 이들 외부 클래스에 대하여 반응하는 클래스의 책임들을 이해하게 되는 복잡성을 고려하였기 때문이다.

6. 결론 및 향후 연구과제

본 논문에서는 UP의 분석단계에서 생성되는 분석 클래스에 대한 복잡도를 측정하기 위한 척도를 정의하였다. 분석 단계에서 시스템의 복잡도를 줄임으로써 개발 과정에 소요되는 시간과 노력을 크게 감소시킬 수 있으므로 분석 클래스에 대한 복잡도 측정은 필수적이다. 그러나, 기존에 연구되었던 척도들이 분석 단계에 나타나는 것보다 훨씬 더 상세한 정보를 요구하였기 때문에, 실제로 객체지향 분석 척도라고 말할 수 없었던 것이 사실이다. 또한, 설계 척도의 일부는 구현이후의 소스 코드에 대하여 측량할 수 있는 것이었다. 이러한 문제를 개선하기 위하여, 제안된 척도는 분석 초기의 클래스에 대한 정보에 기반을 두고 정의되었다. 협력의 복잡도와 인터페이스 복잡도는 분석 클래스의 책임과 협력관계 및 상속 관계를 기반으로 하였기 때문에 분석 단계에서 시스템의 복잡도를 평가해 볼 수 있게된다. 또한, 기존의 척도들이 수학적 명세를 사용하지 않고, 척도의 적용 절차와 가설에 대한 자세한 언급이 없었기 때문에 실제 측정하기가 어려웠던 문제점을 개선하여 본 논문에서 제안하고 있는 복잡도 척도에 대한 자세한 가설과 적용 절차에 대하여 설명하였으며, 척도에 대한 명세를 수학적 공식을 사용하여 이해하기 쉽도록 하였다.

본 논문에서 제안한 복잡도 척도는 Weyuker가 정의한 9가지 공리적 성질에 대하여 객체지향 척도로서의 성질로서 부적합한 성질 9를 제외한 나머지 모두를 만족하고 있음을 보였다. 또한, RUP에 따라 개발된 시스템의 분석 클래스에 대하여 제안된 척도와 기존의 복잡도 척도를 사용하여 복잡도를 측정하였다. CC와 CBO, IC와 WMC의 값을 비교해 본 결과 제안된 복잡도 척도의 계산 결과 값이 그 값들보다 좀 더 복잡도를 잘 표현하고 있었다.

이로써 소프트웨어 개발 주기의 초기에 클래스에 대한 복잡도를 평가해 보고, 나머지 단계에 필요한 시간과 노력을 예측할 수 있는 기초를 제공함으로써 보다 비용-효과적인 객체지향 소프트웨어를 개발할 수 있는 가능성이 높아진다고 할 수 있다.

그러나, 복잡도에 대한 평가와 함께 품질과 어떤 관련이 있는지에 대한 연구가 이루어져야 하며, 척도에 필요한 자료를 수집하고 복잡도를 계산하는 자동화 도구가 현재 개발중이다.

참 고 문 헌

[1] Clark A., Michael S., "Object-Oriented Software Measures," CMU/SEI 95-TR-002, April 1995.

[2] D. H. Abbott, T. D. Korson, and J. D. McGregor, "A Proposed Design Complexity Metric for Object-Oriented Development," Technical Report TR-105, Clemson University, 1994.

[3] G. Booch, J. Rumbaugh, and I. Jacobson, The Unified Modeling Language User Guide, Addison-Wesley Longman, 1998.

[4] G. Booch, J. Rumbaugh, and I. Jacobson, The Unified Software Development Process, Addison-Wesley Longman, 1998.

[5] Ivan P. Paltor, Johan L., "Digital Sound Recorder : A case study on designing systems using the UML notation," TUCS Technical Report No.234, January 1999.

[6] Chidamber S. R., C.F. Kemerer, "A Metrics Suite for Object-Oriented Design," IEEE Trans. Software Engineering, Vol.20, No.6, pp.476-493, June 1994.

[7] Li, Wei and S. Henry, "Maintenance Metrics for the Object-Oriented Paradigm," Proceedings of First International Software Metrics Symposium, IEEE Computer Society Press, pp.52-60, 1993.

[8] R. Sharble, S. Cohen, "The Object-Oriented Brewery : A Comparison of Two Object-Oriented Development Methods," SIGSOFT Software Engineering Notes, Vol.18, No.4, pp.60-73, 1993.

[9] 김유경, 박재년, "객체지향 설계의 특성을 고려한 품질 평가 메트릭스", 한국정보처리학회 논문지, 제7권, 제2호, 2000.

[10] 이종석, 우치수, "객체지향 시스템에서의 클래스 응집도와 결합도 메트릭", 한국정보과학회 논문지, 제27권 제6호, pp.595-606, 2000.

[11] 김은미, 전형수, 장옥배, S. KUSUMOTOD, T. KIKUNO, Y. TAKADA, "C++ 프로그램의 복잡도 척도 및 평가 도구", 정보과학회 논문지, 제3권 제6호, pp.656-665, 1997.

[12] R. Wirfs-Brock, B. Wilkerson, and L. Weiner, "Designing Object-Oriented Software," Prentice-Hall, 1990.

[13] E. J. Weyuker, "Evaluating software complexity measures," IEEE Tran. on SE, Vol.14 No.9, pp.1357-1365, 1988.

[14] J. C. Cherniavsky and C. H. Smith, "On Weyuker's axioms for software complexity measures," IEEE Tran. on SE, Vol.17 No.6, pp.636-638, June 1991.

[15] B. Henderson S., Object-Oriented Metrics : Measures of Complexity, Prentice-Hall, 1996.

[16] 채홍석, 권용래, 배두환, "객체지향 시스템의 클래스에 대한 응집도", 정보과학회 논문지, 제26권, 제9호, pp.1095-1104, 1999.

김 유 경

e-mail : ykkim@cs.sookmyung.ac.kr

1991년 숙명여자대학교 수학과 졸업(학사)

1994년 숙명여자대학교 대학원 전산학과 (이학석사)

1995년~현재 안양대학교 컴퓨터학과 강사

1996년~현재 숙명여자대학교 전산학과 강사

1997년~현재 숙명여자대학교 대학원 전산학과 박사과정 수료

관심분야 : 객체지향 모델링, 소프트웨어 품질 평가, 분산객체 시스템

박 재 년

e-mail : jnpark@cs.sookmyung.ac.kr

1966년 고려대학교 졸업(학사)

1969년 고려대학교 석사

1981년 고려대학교 박사

1979년~1989년 전남대학교 전산통계학과 교수

1983년~현재 숙명여자대학교 정보과학부 교수

관심분야 : 시스템 개발방법론, 품질 평가, 메타 DB, 시뮬레이션