퍼지 및 지능시스템학회 논문지 2005, Vol. 15, No. 6, pp. 780-787

UML 클래스 다이어그램 분석에 의한 객체지향 시스템의 복잡도 연구

The Complexity of Object-Oriented Systems by Analyzing the Class Diagram of UML

정 홍, 김태식

Hong Chung and Taesik Kim

계명대학교 정보통신대학 컴퓨터공학과

요 약

객체지향 시스템의 복잡성 척도에 대하여 많은 연구와 검증이 이루어져 왔다. 대부분의 척도들은 시스템의 부분적 측면, 예를 들어, 객체 간 결합도, 상속 구조의 복잡도, 메소드의 응집도 등에 대한 측정을 목표로 하고 있다. 그런데 소프트웨어 실무자들은 부분적이 아닌 시스템의 전반적인 복잡도를 측정할 수 있기를 바라고 있다. 본 논문은 UML의 클래스 다이어그램을 분석함으로써 객체지향 시스템의 전체적 구조에 대한 복잡도를 연구한 것이다. 클래스 다이어그램은 클래스와 클래스 간 관계로 구성되어 있다. 관계에는 연관 관계, 일반화 관계, 집합 관계 등 세 가지가 있는데, 이 관계들이 객체지향 시스템의 구조를 이해하기 어렵게 하고 있다. 본 연구에서는 이 세 가지 관계를 통합하여 객체지향 시스템의 복잡도를 측정하는 경험적 척도를 제안하고 있다. 이 척도는 소프트웨어 개발자가 코딩하기 전에 객체지향 시스템의 복잡도를 평가해 보고 필요시 설계를 수정할 수 있도록 하게 함으로써 설계 업무에 많은 도움을 줄 것이다.

Abstract

Many researches and validations for the complexity metrics of the object-oriented systems have been studied. Most of them are aimed for the measurement of the partial aspects of the systems, for example, the coupling between objects, the complexity of inheritance structures, the cohesion of methods, and so on. But the software practitioners want to measure the complexity of overall system, not partial. We studied the complexity of the overall structures of object-oriented systems by analyzing the class diagram of UML. The class diagram is composed of classes and their relations. There are three kinds of relations, association, generalization, and aggregation, which are making the structure of object-oriented systems to be difficult to understand. We proposed a heuristic metric to measure the complexity of object-oriented systems by putting together the three kinds of the relations. To analyze the complexity of the structure of a object-oriented system for the maintainability of the system, we measured the degree of understandability of it, the reverse engineering time to draw a class diagram from the source codes, and the number of errors in the diagram. The results of this experiment shows that our proposed metric has a considerable relationship with the complexity of object-oriented systems. The metric will be helpful to the software developers for their designing tasks by evaluating the complexity of the structures of object-oriented systems and redesigning tasks of them for the future maintainability.

Key words : 객체지향 척도, 복잡성 척도, 클래스 다이어그램

1. 서 론

소프트웨어를 개발하는 데 있어서 구조적 패러다임이 객체지향 패러다임으로 대체가 되고 있는 상황에서 객체지향 시스템의 복잡도를 측정하기 위한 많은 연구가 있어왔고 [1,3,5,8,13,14], 제안된 척도를 검증하기 위해 더 많은 실험적 연구가 있어왔다[2,4,6,7,9,10,11,15]. 대표적인 몇 가지의 연구에 대해 살펴보면, Chidamber and Kermerer[5]는 객체지향 시스템의 복잡도 측정에 가장 실질적으로 적용될만한 여섯 개의 척도를 제안하였으며, 현재 가장 많은 실증적 연구[2,6,9,10,15]가 뒤따르고 있다. Li[13]는 Chidamber and Kermerer의 척도에 대한 미비점을 지적하고 이를 보완한 새로운 여섯 개의 척도를 제안하고 있다. Sheldon 등[14]은 Li의 연구를 기반으로 하여 클래스 상속 구조의 유지보수를 위한 상속 계층의 이해성과 변경성을 측정하기 위한 척도를 제안하고 있다. Abreu[1]는 객체지향 시스템의 캡슐화, 상속성, 결합도, 다형성에 대한 여섯 개의 복잡한 척도를 제안했는데, 이 또한 실험적 검증[11]이 진행되고 있다. 그리고 Briand 등[3]은 객체지향 시스템의 결합도 측정을 위한 통합 프레임워크를 제안하고 있으며, Ferneley[8]는 제어흐름의 측정을 중심으로 한 결합성 척도를 제안했다. 그런데, 이들 대부분의 척도들은 객체지향 시스템의 부분적 측정, 예를 들어 객체 간의 결합성, 상속 구조의 복잡도, 메소드의 응집성 등 여러 측면을 별개의 척도로 제안하고 있는 데, 실제 실무 현장에서는 시스템의 복잡도를 종합적으로 판단할 수 있는 통합적 척도를 요구하고 있다.

접수일자 : 2005년 9월 28일
완료일자 : 2005년 11월 4일

본 연구에서는 객체지향 시스템의 분석 및 설계를 위한 표기법 중에서 가장 널리 사용되고 거의 표준화가 되어 있는 UML(Unified Modeling Language)에서 시스템의 정적인 구조 설계를 하는데 사용되는 클래스 다이어그램을 분석하여 객체지향 시스템의 정적 구조 복잡도를 종합적으로 측정하고자 한다. 시스템을 구현하기 전 단계인 설계 단계에서 복잡도를 측정해 봄으로써 개발 중인 시스템을 재평가 하고 필요시 재설계를 함으로써 구현의 오류를 줄임과 함께 개발 비용을 줄일 수 있을 뿐만 아니라 유지보수 시에도 시스템을 쉽게 변경 내지는 개선을 할 수 있도록 할 수 있다.

유지보수성을 위한 시스템의 복잡도를 분석함에 있어 소스코드로부터 역공학에 의한 클래스 다이어그램을 작성해봄으로써 역공학 시간, 오류의 수, 시스템 이해도 등을 측정하고, 이로부터 개발 시 작성한 클래스 다이어그램의 복잡도를 계산하여 구현하였을 시 유지보수성의 난이도를 유추하고자 한다. 그리고 제안한 척도가 객체지향 시스템의 복잡도와 상당한 관계가 있음을 보이고자 한다.

2. 관련 연구의 분석

지금까지 연구된 많은 척도들 중 본 논문과 많은 관련이 있는 Chidamber and Kermerer[5]의 척도, Li[13]의 척도, Sheldon 등[14]이 제안한 척도들을 분석해 보고자 한다.

Chidamber and Kermerer는 객체지향 시스템의 복잡도 측정에 관한 여섯 개의 척도를 제안하고 있다. 클래스 상속 트리 구조의 깊이를 측정하는 DIT(Depth of Inheritance Tree)는 "클래스의 상속 깊이"로 정의한다. 이는 클래스에 영향을 미치는 조상 클래스의 수를 측정하는 것인데, 다중 상속인 경우 측정에 모호한 점이 발견된다. 클래스 상속 구조의 너비를 측정하는 NOC(Number Of Children)는 "클래스에 직접 부속된 서브클래스의 수"로 정의한다. 이는 클래스가 상속에 의하여 영향을 미치는 서브클래스의 범위를 측정하는 것인데, 왜 직접 부속된 서브클래스만 측정하는지 분명하지 않다. 클래스의 메소드 수를 측정하는 WMC(Weighted Methods per Class)는 "클래스의 지역적 메소드의 복잡도의 합"으로 정의한다. 각 메소드의 복잡도는 구조적 패러다임에서 사용하는 LOC(Lines Of Codes)나 Cyclomatic Complexity를 가중치로 사용할 수 있다. 시스템 설계 수준에서는 각 메소드의 복잡도를 측정할 수가 없으므로 가중치를 단위 값인 1로 취하여 클래스의 메소드의 수를 클래스의 복잡도로 간주할 수 있다. 객체 간의 동적 행위인 메시지 전달의 수를 측정하는 RFC(Response For Class)는 "클래스의 객체가 받은 메시지에 대한 응답으로 수행될 가능성이 있는 메소드의 수"로 정의한다. 클래스 간의 결합도를 측정하는 CBO(Coupling Between Objects)는 "결합이 되는 다른 클래스의 수"로 정의한다. 클래스 내의 메소드와 인스턴스 변수 간의 결합성을 측정하는 LCOM(Lack of COhesion in Methods)은 "클래스의 인스턴스 변수 집합의 쌍의 수"로 정의한다. 정의에서는 인스턴스 변수 집합의 쌍이라 하고 있으나 모호성이 있어 실제로는 메소드 집합의 쌍으로 보아야 한다. 이상과 같이 여섯 개의 척도 중 DIT와 NOC는 클래스 상속 구조에 관한 것이고, WMC, RFC, LCOM은 클래스의 복잡도에 관한 것이며, CBO는 클래스 간 연관관계에 관한 것이다.

Li[13]도 객체지향 시스템의 복잡도에 관한 여섯 개의 척도를 제안하고 있다. 클래스 상속 계층의 복잡도를 측정하는 NAC(Number of Ancestor Classes)는 "클래스가 상속받는 조상 클래스의 수"로 정의한다. 클래스의 상속 계층은 유향 그래프로 표현할 수 있는데, 이 척도는 유향 그래프에서 특정 노드에 도달될 수 있는 노드의 수이다. 즉, 이는 특정 클래스에 영향을 미치는 모든 조상 클래스의 수를 측정하는 것이다. 클래스의 상속 계층의 복잡도를 측정하는 NDC(Number of Descendent Classes)는 "클래스가 상속하는 자손 클래스의 총수"로 정의한다. 상속 계층의 유향 그래프에서 특정 노드로부터 도달할 수 있는 노드의 수이다. 즉, 이는 특정 클래스로부터 영향을 받는 모든 자손 클래스의 수를 측정하는 것이다. 클래스의 복잡도를 측정하는 NLM(Number of Local Methods)은 "클래스 외부에서 접근할 수 있는 지역 메소드의 수"로 정의한다. 지역 메소드란 클래스 내에서 public으로 선언된 메소드를 말한다. 이 척도는 다른 클래스가 이 클래스를 사용할 수 있도록 하는 지역 인터페이스의 크기를 표시한다고 볼 수 있다. 클래스의 복잡도를 측정하는 CMC(Class Method Complexity)는 "외부 클래스에서 가시적이든 아니든(public or private) 모든 지역 메소드의 내부구조의 복잡도의 합"으로 정의한다. 메소드의 복잡도는 LOC나 Cyclomatic Complexity가 될 수 있다. 클래스 간의 결합도를 측정하는 CTA(Coupling Through ADT)는 "클래스의 데이터 속성 선언에서 ADT로 사용되는 클래스의 수"로 정의한다. 이 척도는 클래스가 다른 클래스에 서비스를 제공하기 위해 얼마나 많은 다른 클래스의 서비스가 필요한가를 나타낸다. 객체 간 동적 결합도를 측정하는 CTM(Coupling Through Message passing)은 "클래스의 지역 메소드에서 지역 객체로 생성된 객체를 제외하고, 클래스에서 다른 클래스로 보내는 여러 다른 메시지의 수"로 정의한다. 이 척도는 클래스가 자신의 기능을 수행하기 위해 얼마나 많은 다른 클래스의 메소드가 필요한가를 나타낸다. 이상과 같이 여섯 개의 척도 중 NAC와 NDC는 클래스 상속 구조에 관한 것이고, NLM과 CMC는 클래스의 복잡도에 관한 것이며, CTA와 CTM은 클래스 간의 연관관계에 대한 것이다.

Sheldon 등[14]은 클래스 상속 구조의 복잡도에 관한 두 개의 척도를 제안하고 있다. 클래스 상속 계층의 이해도를 측정하기 위한 U(Understandability)는 "클래스에 영향을 주는 조상 클래스의 수"로 정의하고 있는데, 유향 그래프로 보았을 때 선행자 수를 계산한다. 클래스 상속 계층의 변경성을 측정하기 위한 M(Modifiability)은 "클래스의 이해도와 그 클래스에 영향을 미치는 자손 클래스의 수의 합"으로 정의하고 있는데, 유향 그래프로 보았을 때 후속자 수를 계산한다. 영향을 미친다는 것은 자손 클래스의 변경이 일어나는 것인데, 경험 법칙에 의해 반 정도 변경이 일어난다고 보고 후속자 수를 2로 나누고 있다. 그리고 각 클래스의 U와 M 외에 전체 상속 계층에 대한 합계인 TU(Total Understandability)와 TM(Total Modifiability)을 제안하고, 또한 이를 클래스 수로 나눈 평균 값인 AU(Average Understandability)와 AM(Average Modifiability)을 제안하고 있다. 이 척도들은 모두 클래스 상속 구조의 복잡도에 관한 것이다.

이상 객체지향 시스템의 복잡도에 관해 제안된 몇 가지 척도들을 살펴보았는데, 모두 객체지향 시스템의 일부 구조의 복잡도에 관한 것들을 서로 독립적으로 정의하고 있다. 전반적인 시스템의 복잡도는 실제 여러 요소들이 복합적으로 연계하여 발생한다. 예를 들어 객체지향 시스템의 정적 구조를 설계한 클래스 다이어그램에는 클래스 간에 상속관계, 연

관관계, 집합관계 등이 동시에 나타나는 것이 보통이므로 전체적인 시스템의 복잡도는 이들 요소의 복잡도를 결합하여 계산해야 한다.

3. 클래스 다이어그램의 복잡도 분석

클래스 다이어그램으로 표현한 시스템의 정적 구조의 복잡도는 클래스 간의 관계가 얼마나 복잡한가에 의하여 결정된다. 클래스 간의 관계에는 연관(association) 관계, 집합(aggregation) 관계, 일반화(generalization) 관계가 있다. UML(Unified Modeling Language)에서는 그림 1과 같이 연관관계는 단방향(unidirection)인 경우는 연관되는 방향으로 실선 화살표, 양방향인 경우는 실선으로 나타내고, 일반화 관계는 상속하는 클래스 쪽에 작은 삼각형을 표시한 실선, 집합 관계는 집합 클래스 쪽에 작은 마름모꼴을 표시한 실선으로 나타낸다.

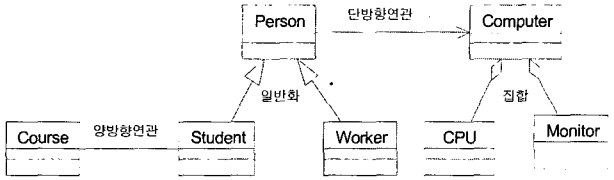

그림 1. 클래스 다이어그램
Fig. 1. Class Diagram

3.1 연관 관계

연관관계는 클래스들 사이에 가장 광범위한 관계로서 상대 객체에 대한 인지를 나타낸다. 일방적으로만 인지하는 단방향 연관과 서로가 인지하는 양방향 연관이 있다. 그림 1의 Person 클래스와 Computer 클래스의 단방향 연관에 대한 코딩 예를 보면 다음과 같다.

```
class Person {
        private Computer theComputer;
        private int value;
        .....
        public string CalculateValue() {
                value = theComputer.Calculate();
        }
        .....
}

class Computer {
        .....
        public int Calculate() {...}
        .....
}
```

Person 클래스는 Computer 클래스의 인스턴스인 theComputer 객체를 선언하여 이 객체에 Calculate() 메시지를 보낸다. Person 클래스에 theComputer 객체를 선언한다는 것은 Person 클래스가 Computer 클래스의 내용을 잘 알고 있어야 한다는 것이다.

그림 1의 Course 클래스와 Student 클래스의 양방향 연관에 대한 코딩 예를 보면 다음과 같다.

```
class Course {
        private string courseId;
        private int credit;
        private int noOfSt;
        private Student theStudent;
        public string NumberOfStudent() {
                noOfSt = theStudent.GetNoOfSt();
        }
        public int getCredit() {...}
        .....
}

class Student {
        private string id;
        private int name;
        private Course theCourse;
        private int totalCredit;
        public int GetNoOfSt() {...}
        public int CalculateCredit() {
                totalCredit = theCourse.getCredit();
        }
        .....
}
```

Course 클래스는 Student 클래스의 인스턴스인 theStudent 객체를 선언하여 이 객체에 GetNoOfSt() 메시지를 보내며, 또한 Student 클래스는 Course 클래스의 인스턴스인 theCourse 객체를 선언하여 이 객체에 getCredit() 메시지를 보낸다. 이는 Course 클래스가 Student 클래스를, 그리고 Student 클래스는 Course 클래스를 잘 알고 있어야 한다는 것을 의미한다.

단방향과 양방향의 연관관계를 복잡도 측면에서 비교해 보면, 단방향 연관관계는 메시지를 보내는 상대방의 클래스 구조에 대해서만 이해하면 되나, 양방향 연관관계는 클래스 서로 간에 상대 클래스를 인지하여야 메시지를 보낼 수 있으므로 두 클래스 모두 상대방 클래스의 구조에 대해 이해를 해야 된다. 따라서 경험 법칙에 의하면 양방향 연관관계의 복잡도는 단방향 연관관계의 복잡도보다 두 배가 복잡한 연관관계라 할 수 있다.

3.2 일반화 관계

일반화 관계는 클래스 간의 상속을 통한 개념의 일반화를 나타내는데, 서브클래스는 슈퍼클래스의 속성과 연산을 상속받으므로 객체지향 언어에서는 상속 관계라고도 한다. 슈퍼클래스는 서브클래스들의 공통 성질을 일반화한 클래스이고, 반대로 서브클래스들은 슈퍼클래스의 일반화된 성질을 상속받음과 동시에 각 서브클래스 고유의 성질을 추가한 특수화 클래스이므로 일반화/특수화 관계라고도 한다. 그림 1의 슈퍼클래스 Person과 서브클래스 Student와 Worker의 상속관계에 대한 코딩 예를 보면 다음과 같다.

```
class Person {
        private string name;
        private int age;
        public string GetName() {...}
        public int HowOld() {...}
}
```

```
class Student : Person {
        private string school;
        private int grade;
        public string WhatSchool() {...}
        public int WhatGrade() {...}
}

class Worker : Person {
        private string company;
        private int salary;
        public string WhatCpmpanyl() {...}
        public int GetSalary() {...}
}
```

Person 클래스는 Student 클래스와 Worker 클래스 모두가 가지는 공통 속성과 연산을 가지고 있으며, Student 클래스와 Worker 클래스는 Person 클래스의 속성과 연산을 포함하면서 각기 고유의 속성과 연산을 추가로 선언하여 사용하고 있다.

상속 관계는 클래스의 공통 사항은 슈퍼클래스에 코딩하고 서브클래스에는 그 클래스에만 추가로 필요한 사항만 코딩하므로 코딩의 중복을 줄여주는 재사용의 장점이 있으나, 클래스 구조의 이해성 측면에서 볼 때 서브클래스인 Student나 Worker의 구조를 이해하려면 슈퍼클래스 Person을 찾아 이해를 해야 하는 복잡성을 띄게 된다.

상속구조는 이해성(understandability) 측면에서 복잡도를 측정할 수 있다. 즉, 상속구조의 이해가 어려우면 복잡도가 높으며, 이해가 쉬우면 복잡도가 낮다는 경험 법칙에 의한 것인데, 이를 위해서는 Sheldon 등[14]의 이해도인 U, TU, AU를 그대로 사용할 수가 있다.

3.3 집합 관계

집합 관계는 연관 관계의 특수한 경우로서, 한 클래스의 객체가 다른 클래스의 객체를 부분으로서 포함하는 것을 나타낸다. 때로는 집합관계를 적용할지 연관관계를 적용할지 애매한 경우가 많은데, 중요한 차이점은 부분적 의미의 적용이 집합관계는 가능하나 연관관계에서는 불가능하다. 또한 연관관계는 양방향 연관관계처럼 대칭성이 가능하나 집합관계는 비대칭적이다. 그림 1의 Computer 클래스와 CPU 및 Monitor 클래스의 집합관계에 대한 코딩 예를 보면 다음과 같다.

```
class Computer {
        private int memorySize;
        private CPU theCPU;
        private Monitor theMonitor
        .....
}

class CPU {...}

class Monitor {...}
```

Computer 클래스는 CPU 클래스, Monitor 클래스의 집합으로 이루어져 있으므로, Computer 클레스는 집합을 이루는 두 클래스 모두를 잘 알아야 한다.

집합관계는 클래스 구조상 단방향 연관관계로 볼 수 있는데 경험 법칙에 의하면 이해하기가 쉬워 복잡도가 낮다고 볼 수 있다.

4. 객체지향 시스템의 정적 구조 복잡성 척도

객체지향 시스템의 정적 구조 복잡도는 클래스나 객체 간의 결합의 수로써 측정하고 있다. Chidamber and Kermerer[5]는 객체 간의 결합의 수를 측정하는 CBO를 제안하여 이것이 많을수록 시스템의 구조가 복잡하다고 하고 있으며, Henderson-Sellers[12]는 클래스 간 fan-in/fan-out의 수를 측정하여 시스템 구조의 복잡도로 사용하고 있다. CBO는 결합의 방향성을 고려하지 않고 단순히 결합의 수만 측정하고 있으나 fan-in/fan-out은 방향성을 고려하여 결합의 수를 측정하고 있다. 양방향의 결합은 fan-in과 fan-out을 동시에 가짐으로 복잡도 측면에서 볼 때 두 배로 복잡하다고 해야 할 것이다. 따라서 본 연구에서는 클래스 간 결합을 fan-in이나 fan-out의 하나에 해당하는 단방향과 fan-in과 fan-out 모두를 가지는 양방향으로 구분하여 복잡도를 계산하고자 한다.

4.1 종합적 복잡도 분석에 의한 척도의 제안

객체지향 시스템의 정적 구조를 나타내는 클래스 다이어그램에서 클래스 간 연관관계, 일반화관계, 집합관계에 의해 시스템의 복잡도를 종합적으로 분석하고자 한다.

본 연구에서는 객체지향 시스템의 정적 구조 즉, 클래스 다이어그램의 복잡도를 클래스 간 관계의 수로써 정의한다. 왜냐하면 클래스 다이어그램의 복잡성은 클래스 간에 얼마나 많은 관계가 존재하는가에 따라 파악되는 것이 일반적이기 때문이다. 이 때 연관관계와 집합관계는 직접 연결된 관계의 수를 파악하고, 상속관계는 직상위 혹은 직하위 상속관계 뿐 아니라 모든 조상 클래스와의 관계 및 모든 자손 클래스까지의 관계를 포함한다.

복잡도 측정을 위한 척도의 정의는 시스템 분석 시 적용하고자 하는 경험 법칙과 함께 다음과 같이 전개한다.

1) 연관관계는 단방향과 양방향 연관관계가 있는 바, 양방향 연관관계가 단방향 연관관계보다 2배 복잡한 것으로 간주한다. 왜냐하면 단방향 연관관계는 한 클래스가 연관되는 상대 클래스만 인지하면 되나, 양방향 연관관계는 상대 클래스도 본 클래스를 인지해야 하기 때문이다. 여기서 인지라 함은 구현 시 메시지를 보내려면 상대 클래스의 인스턴스(instance)인 객체의 구조와 제공하는 연산을 모두 알아야 함을 의미한다. 따라서 연관관계의 복잡도 AR(Association Relationship)은 다음과 같다.

AR=(단방향 연관관계 수)+2*(양방향 연관관계 수) (1)

2) 일반화 관계의 복잡도는 Sheldon 등[14]이 제안한 다음과 같은 평균 이해도 AU를 적용한다. AU의 계산에 있어서 모든 클래스의 내부 복잡도는 모두 똑 같은 단위 값 1로 간주한다. 즉, 클래스의 수 혹은 상속관계의 수로 한다.

$$AU=(\sum_{i=1}^{n}(PRED(Ci)+1))/n \qquad (2)$$

여기서 n은 클래스 다이어그램에 있는 클래스의 수이며, PRED(Ci)는 클래스 Ci의 선행자의 수이다. 상속 구조에서 클래스 Ci를 이해하려면 그 자신 뿐 아니라 자신에 영향을 주는 모든 조상 클래스 즉, 상속 구조를 유향 그래프로 표현했을 때의 선행자 수만큼의 클래스도 이해를 해야 한다. 그리고 클래스마다의 복잡도가 다르므로 평균을 취한다.

3) 집합관계의 복잡도는 단방향 연관관계와 동일하다고 가정한다. 왜냐하면 집합관계는 여러 클래스가 모여 하나의 클래스를 이루는 형태이므로, 집합된 하나의 클래스는 모든 멤버 클래스를 인지해야 하나, 멤버 클래스들은 집합 클래스를 인지할 필요가 없기 때문이다.

AGR(AGgregation Relationship)=(집합관계의 수) (3)

척도 이론에 의하면 식 (1), (2), (3)은 단위가 모두 클래스 간 관계의 수이므로 가법의 성질을 갖는다. 따라서 클래스 다이어그램의 복잡도 CCD를 종합적으로 표시하려면 다음과 같이 더할 수 있다.

$$CCD = AR + AU + AGR \quad (4)$$

식 (4)를 식 (1), (2), (3)으로 대입하면 클래스 다이어그램의 복잡도는 다음과 같다.

CCD={(단방향 연관관계 수)+2*(양방향 연관관계 수)}

$$+(\sum_{i=1}^{n}(PRED(Ci)+1))/n+(\text{집합관계의 수}) \quad (5)$$

식 (5)에 의해 그림 1의 복잡도를 계산해보면 다음과 같다.

$$CCD = 1+2*1+(1+2+2)/3+2 = 6.67$$

따라서 그림 1의 클래스 다이어그램은 복잡도는 매우 낮은 편이다. 여러 객체지향 시스템 개발 프로젝트에 참여한 대학생들의 시스템 설계 수준으로 볼 때 CCD가 30 이하이면 시스템의 구조가 간단한 편이고 30 이상이 되면 상당히 복잡한 것으로 생각된다. 30이라는 정량화된 복잡도 수준은 본 연구자의 경험적 실험에 의한 것으로서, 다음 절에서 복잡도 측정 실험을 한다.

4.2 복잡도 측정 실험

본 연구에서 실험하고자 하는 대상은 대학정보시스템의 일부로서 사용자관리를 포함한 학생관리 기능과 수강신청 기능에 대한 것이다. 이 시스템은 객체지향 시스템으로 Java로 코딩이 되어 있다.

실험에 참여하는 학생은 객체지향시스템과 UML을 배운 학생으로서 성적이 B이상인 24명의 학부학생과 대학원생 6명을 참여시켰다. 24명의 학부학생은 각각 4명씩 6개 그룹으로 나누고 각각의 그룹에 대학원생 1명씩을 포함시켜 6개 측정 그룹 군을 구성하였다. 측정하고자 하는 내용은 시스템을 이해하는 이해도와 소스코드를 역공학하여 클래스 다이어그램으로 나타내는데 걸리는 시간을 조사하고 또한 클래스 다이어그램으로 나타냈을 때 오류의 수를 파악하는 것이다. 그리고 본 논문에서 제안한 복잡성 척도와 이들 간의 관계를 알아보고자 한다.

4.2.1 실험하고자 하는 시스템의 구성

실험하고자 하는 대상은 대학정보시스템의 일부인데, 이의 구조를 UML 패키지 다이어그램으로 표현하면 그림 2와 같다.

본 연구에서는 실험 대상을 다음과 같이 6가지 경우로 나누어 복잡도를 조사한다

경우 A: 사용자관리
경우 B: 학생관리
경우 C: 수강신청관리
경우 D: 사용자관리 + 학생관리
경우 E: 사용자관리 + 수강신청관리
경우 F: 사용자관리 + 학생관리 + 수강신청관리

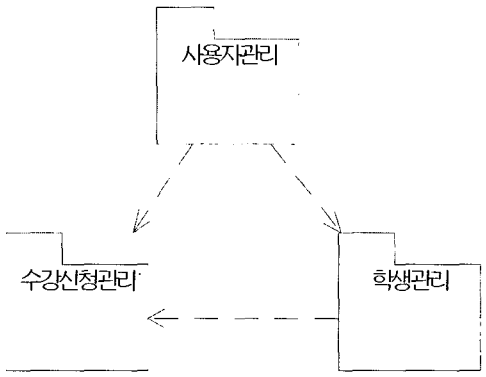

그림 2. Package Diagram
Fig. 2. 패키지 다이어그램

6가지 경우에 대한 클래스 다이어그램을 본 연구에서 제안한 척도로 복잡도를 계산하면 표 1과 같다.

표 1. 클래스 다이어그램의 복잡도
Table 1. Complexity of the Class Diagram

	AR	AU	AGR	CCD
경우 A	16	2	0	18
경우 B	16	1.5	1	18.5
경우 C	15	1.5	2	18.5
경우 D	30	2	1	33
경우 E	30	2	2	34
경우 F	39	2	3	44

4.2.2 측정하고자 하는 내용

측정하고자 하는 내용은 시스템을 이해하는 심리적 어려움을 나타내는 이해도와 소스코드를 역공학하여 클래스 다이어그램으로 나타내는데 걸리는 시간, 그리고 클래스 다이어그램으로 나타냈을 때 오류의 수이다.

이해도는 다음과 같이 5레벨로 한다.

레벨 1: 매우 쉬움
레벨 2: 쉬움
레벨 3: 보통
레벨 4: 어려움
레벨 5: 매우 어려움

역공학 소요 시간은 분단위로서 1분으로부터 300분까지의 시간이다. 50분 작업 뒤 10분간씩 휴식을 하도록 했으며, 휴식 시간에는 서로 간에 의사소통을 못하게 했다. 오류의 수는 측정 그룹별로 역공학하여 작성한 클래스 다이어그램과 개발 시 문서화된 클래스 다이어그램과 비교하여 발견된 오류를 계산한다.

4.2.3 실험 결과 분석

6가지 경우에 대해 각 경우를 6개 측정 그룹 군에 각각 할당하여 실험을 수행했는데. 그 결과는 표 2와 같다.

여기서 학부생에 대한 측정치는 4명의 평균 값이다. 그리고 각 측정 그룹별 평균은 대학원생과 학부생의 측정치를 평

균함으로써 잘하는 사람과 못하는 사람의 평균값을 취한 것이다.

표 2. 실험 결과
Table 2. Results of the Experiment

경우	CCD	참여자	이해도 수준	소요 시간	오류 수
A	18	대학원생	1	64	0
		학부생	1	104	0
		평 균	1	84	0
B	18.5	대학원생	1	65	0
		학부생	1.6	122	1.4
		평 균	1.3	94	0.7
C	18.5	대학원생	1	71	0
		학부생	2.3	109	2.2
		평 균	1.7	90	1.1
D	33	대학원생	2	156	1
		학부생	3.3	220	3.5
		평 균	2.7	188	2.3
E	34	대학원생	2	135	2
		학부생	3.5	212	3.4
		평 균	2.8	174	2.7
F	44	대학원생	4	283	4
		학부생	5	300	8.6
		평 균	4.5	292	6.3

복잡도의 증가에 따른 이해도, 소요시간, 오류의 수에 대한 변화를 그래프로 나타내면 그림 3 및 그림 4와 같다.

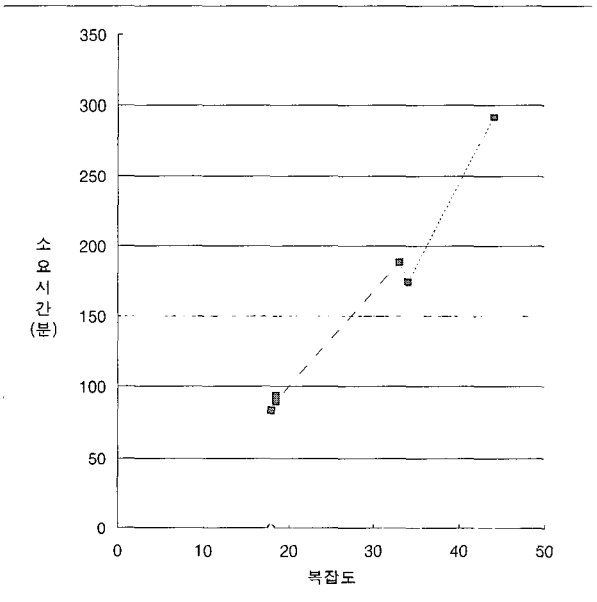

그림 3. 복잡도에 따른 소요시간의 증가
Fig. 3. Increment of Elapsed Time with the Complexity

본 그래프를 분석해 보면 복잡도가 증가할수록 각 측정치가 급하게 증가함을 볼 수가 있다. 3가지 측정치에 대한 복잡도가 18 이하인 시스템에 대해서는 실험을 하지 못했으나 경험상 상당히 간단한 구조일 것이므로 측정치도 적으면서 완만하게 변화할 것으로 판단된다. 그리고 본 실험 결과에서 경험적으로 판단해 보건데 복잡도가 30정도인 경우의 이해도가 대략 보통이라고 응답하고 있으므로, 시스템 설계 시 하나의 클래스 다이어그램의 복잡도는 30 이하로 하는 것이 바람직하다고 볼 수 있다.

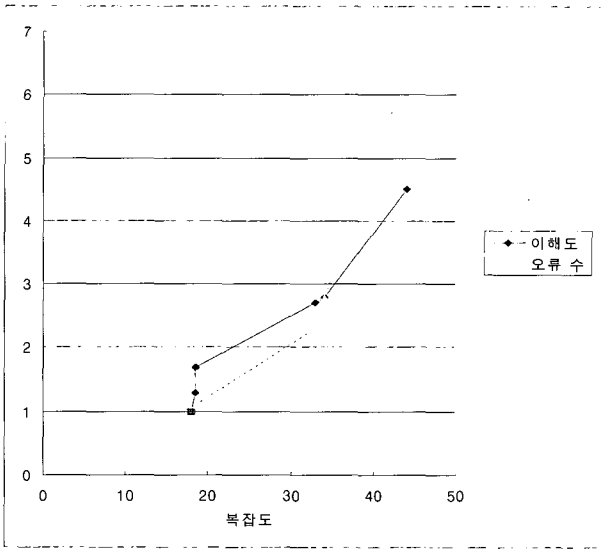

그림 4. 복잡도에 따른 이해도와 오류 수의 증가
Fig. 4. Increment of Understandability level and the Number of Errors with the Complexity

4.3 제안한 복잡성 척도의 평가

객체지향 시스템의 설계는 먼저 사용자 요구사항 명세로부터 클래스를 찾아 정의하고 다음에 클래스 간의 여러 가지 관계를 도출하여 시스템의 정적 구조인 클래스 다이어그램을 작성한다. 이 때 작성한 클래스 다이어그램은 차후 객체 설계, 객체 간 메시지 설계 등의 작업에 많은 영향을 미치므로 정확성은 물론이거니와 복잡성에 대해서도 분석 및 고찰을 해보아야 한다. 정확성은 사용자 요구사항과 비교해보아야 함으로 응용 종속적이나 복잡성에 대해서는 클래스 다이어그램의 구조를 보고 객관적, 정량적 판단이 가능하다. 이에 대해 Chidamber and Kermerer[5]의 척도인 DIT와 NOC 및 Li[13]의 척도인 NAC와 NDC, 그리고 Sheldon[14]의 U와 AU는 상속관계만 측정하고 있다. 그리고 Li[13]의 CTA와 Abreu[1]의 결합도는 연관관계만 측정하고 있다. 이와 같이 제안된 대부분의 척도들은 시스템의 구조 복잡도를 요소별로 측정하도록 하여 요소별 복잡도만 파악할 뿐이며 전체적인 구조 복잡도는 종합적으로 판단하기가 어렵다. 본 연구에서는 이들 요소(연관관계, 상속관계, 집합관계)를 경험 법칙에 의한 방법으로 시스템의 구조 복잡도를 통합적으로 판단할 수 있는 척도를 제안하였다.

본 연구에서는 측정 그룹을 대학원생과 학부학생으로 했는데, 실제 소프트웨어 개발 업무에 종사하는 프로그래머를 대상으로 실험을 했다면 좀더 실제적인 값이 측정되었으리라 생각된다. 그리고 실험 대상도 대학정보시스템이 아닌 일반 기업체에서 운용하고 있는 시스템의 소스코드를 사용했더라면 좀더 실질적인 결과를 얻었으리라 생각된다. 그러나 복잡도 수준에 따른 이해도 수준, 역공학 소요시간, 오류 수의 변화 패턴은 유사하리라 생각된다. 그런데 본 척도에 대한 타당성을 확보하기 위해서는 좀더 많은 실험적 평가가 이루어져야 한다. 실험을 위해서는 다양한 숙련도를 가진 시스템 설계자의 참여와 테스트를 위한 다양한 종류와 크기의 시스템을 확보해야 하며, 무엇보다 시스템 설계자가 느끼는 복잡

도 수준을 잘 정하는 것이다. 예를 들어 문제의 난이도를 판단하는데 있어서 사람에 따라 어떻게 난이도를 결정하는지를 검증할 수 있도록 해야 하며, 같은 난이도인 경우에도 실험에 참가한 학생의 형태에 따라 이해도 수준, 역공학 수준, 오류 수에 상당한 차이를 보일 수 있다. 따라서 복잡도가 30 이상이라고 하여 항상 시스템이 복잡하다고 보기 어려운 경우가 있을 것이며, 그 이하라고 해서 항상 간단한 시스템이라고 단정하기 어려운 경우도 있을 것이다. 따라서 복잡도 수준의 정량화는 인간의 인지적 복잡도(cognitive complexity)라는 문제와 연관이 되므로 인지적 복잡도와 연관된 연구가 필요하다.

본 연구의 결과는 시스템의 절대적 복잡도 수준을 결정하기는 매우 어렵겠지만 시스템 개발자가 느끼는 상대적 복잡도 수준은 어느 정도 결정 가능할 것이다.

5. 결 론

객체지향 시스템의 정적 구조 설계 시 시스템 개발자들은 일반적으로 UML 표기법인 객체 다이어그램을 사용한다. 객체지향 시스템의 복잡도 측정을 위한 많은 연구 및 실험적 검증이 있어왔지만 대부분이 단편적인 복잡도의 측정에 불과해 개발자에게 별로 도움을 주지 못하고 있다.

본 연구는 클래스 다이어그램의 클래스 간의 주요 관계인 연관관계, 일반화관계, 집합관계의 복잡도를 분석하고 이를 통합적으로 적용할 수 있는 복잡성 척도를 제안하고 실 예에 적용해 보았다. 실험 결과로서 본 논문에서 제안한 복잡성 척도가 시스템을 이해하는 정도, 오류의 발생 수, 소스 코드를 역공학 하는데 소요되는 시간과 상당한 연관이 있음을 보였으며, 아직 완전한 타당성이 부족하다고 볼 수 있지만 복잡도가 대략 30을 넘으면 구현이 쉽지 않고 또한 유지보수성이 어려울 것임을 보였다.

본 연구의 결과는 시스템 개발자들이 시스템을 구현하기 전 단계인 설계 단계에서 복잡도를 측정해 봄으로써 개발 중인 시스템을 재평가 하고 필요시 재설계를 함으로써 구현의 오류를 줄임과 함께 개발비용을 줄일 수 있을 뿐만 아니라 유지보수 시에도 시스템을 쉽게 변경 내지는 개선을 할 수 있도록 할 수 있을 것이다.

그런데 본 연구는 경험 법칙에 의한 여러 가정 하에 클래스 다이어그램의 복잡성 척도를 유도한 것이므로 실제 실무에 적용하기 위해서는 더 많은 실험적 평가를 거쳐야 한다. 또한 적합한 복잡도 수준을 정량적으로 결정해야 하는데, 이를 위해서는 소프트웨어에 대한 개발자의 인지적 복잡도라는 문제를 해결해야 한다. 앞으로의 계속적인 연구 방향은 인지적 복잡도 모델의 연구와 복잡성 척도의 실험적 평가이다.

참 고 문 헌

[1] F. Abreu, "The MOOD Metrics Set," *Proc. ECOOP'95 Workshop on Metrics*, 1995

[2] V. Basili, L. Briand, and W. Melo, "A Validation of Object-Oriented Design Metrics as Quality Indicators," *Technical Report,* University of Maryland, Department of Computer Science, College Park MD, pp.1-24, 1995,

[3] L. Briand, J. Daly and J. Wust, "A Unified Framework for Coupling Measurement in Object-Oriented Systems," *IEEE Transactions on Software Engineering,* Vol. 25, No. 1, pp.91-121, 1999

[4] L. Briand, J. Wust, J. Daly, and D. Porter, "Exploring the relationships between design measures and software quality in object-oriented systems," *The Journal of Systems and Software,* Vol 65, No. 3, pp.245-273, 2000

[5] S. Chidamber and C. Kermerer, "A Metrics Suite for Object-Oriented Design," *IEEE Transactions of Software Engineering*, Vol. 20, No. 6, pp.476-493, 1994

[6] I. Deligiannis, M. Shepperd, M. Roumeliotis and I. Stamelos, "An empirical investigation of an object-oriented design heuristic for maintainability," *The Journal of Systems and Software,* Vol 65, pp.127-139, 2003

[7] K. Emam, W. Melo, and J. Machado, "The prediction of faulty classes using object-oriented design metrics," *The Journal of Systems and Software,* Vol. 56, No. 1, pp.63-75, 2001

[8] E. Ferneley, "Coupling and control flow measures in practice," *The Journal of Systems and Software*, Vol. 51, pp.99-109, 2000

[9] Gursaran, "Viewpoint representation validation: a case study on two metrics from the Chidamber and Kemerer suite," *The Journal of Systems and Software,* Vol. 59, No. 1, pp.83-97, 2001

[10] R. Harrison, S. Counsell, and R. Nithi, "Experimental assessment of the effect of inheritance on the maintainability of object-oriented systems," *The Journal of Systems and Software,* Vol. 52, No. 2-3, pp.173-179, 2000

[11] R. Harrison, S. Counsell, and R. Nithi, "An Evaluation of the MOOD Set of Object-Oriented Software Metrics," *IEEE Transactions on Software Engineering*, Vol. 24, No. 6, pp.491-496, 1998

[12] B. Henderson-Sellers, *Object-Oriented metrics : measures of complexity*, Prentice Hall PTR, 1996

[13] W. Li, "Another metric suite for object-oriented programming," *The Journal of Systems and Software,* Vol 44, No.2, pp.155-162, 1998

[14] F. Sheldon, K. Jerath and Hong Chung, "Metrics for maintainability of class inheritance hierarchies," *Journal of Software Maintenance and Evolution*, Vol 14, pp.147-160, 2002

[15] G. Subramanian and W. Corbin, "An empirical study of certain object-oriented software metrics," *The Journal of Systems and Software,* Vol. 59, pp.57-63, 2001

저 자 소 개

정 홍(Hong Chung)
1972년 : 한양대학교 (공학사)
1976년 : 고려대학교 (경영학석사)
1996년 : 대구가톨릭대학교 (이학석사)
1999년 : 대구가톨릭대학교 (이학박사)
1972년~1981년 : 한국과학기술연구원 (KIST) 선임연구원
2000년~2001년 : Washington State University 연구교수
1981년~현재 : 계명대학교 정보통신대학 부교수

관심분야 : 지능정보시스템, 소프트웨어공학
E-mail : jhong@kmu.ac.kr

김태식(Taesik Kim)
1984년 : 계명대학교 컴퓨터공학과(공학사)
1987년 : Minnesota State University(석사)
1992년 : North Dakota State University (박사)
1992년~현재 : 계명대학교 정보통신대학 부교수

관심분야 : Chaos, Neural Network
E-mail : tskim@kmu.ac.kr