객체지향 설계의 특성을 고려한 품질 평가 메트릭스

김 유 경† · 박 재 년††

요 약

지금까지의 객체지향 메트릭스에 대한 연구는 단순히 품질의 일부 요소만을 다루고 있으며, 대부분의 메트릭스는 클래스 사이의 관계 정보만을 기반으로 제안되어 왔다. 이로써 객체지향 설계의 특성을 충분히 반영하지 못하고 있다.

또한, 기존의 객체지향 메트릭스는 각 메트릭의 계산 결과를 평가하기 위하여 제한값(threshold)을 제공하고 있으며, 형식적으로 정의되지 않은 것이 대부분이다. 이들의 문제점은 계산 과정이 복잡하여 쉽게 적용할 수 없고, 프로젝트의 성격이나 소프트웨어의 특성에 따라 제한 값이 달라질 수 있다는 것이다.

이에 본 논문에서는 객체지향 설계의 특성인 크기, 복잡도, 결합도 및 응집도를 고려하여 설계의 품질을 평가하기 위한 메트릭 집합을 제시한다. 제시된 메트릭 집합은 평균값에 대한 비율(proportion)을 사용하여 평균값을 상회하는 클래스 및 설계 요소들을 쉽게 파악 할 수 있도록 하였다. 이것은 설계 품질을 저하시키고 있는 클래스를 찾아내어, 평균값에 근접한 수준으로 다시 설계할 수 있도록 함으로써, 구현하는 동안 직면하는 설계 결점을 개발 초기에 발견할 수 있도록 하였다.

본 논문에서 정의한 메트릭 집합은 측정원칙에 의하여 분석적으로 평가된다. 그 결과로서 메트릭에 대하여 요구되는 성질의 대부분을 만족하고 있음을 알 수 있다. 또한, 플랫폼과 무관하게 사용할 수 있도록 웹 브라우저 및 자바 애플릿으로 개발되어 분산 인트라넷 환경을 지원하는 평가 도구 ASSOD(ASsessment System of Object oriented Design)를 설계한다.

Metrics Measuring a Quality based on Object-Oriented Design Characteristics

Yu-Kyong Kim† · Jai-Nyun Park††

ABSTRACT

There are many researches about metrics to measure a quality of Object-Oriented(OO) software. However, most of them have only discussed a concept or properties of metrics, and have not shown the detailed procedure for measuring them. They also define a measurement indicator as a threshold, but it has been influenced on a project size or application domains.

In this paper, we propose metrics based on characteristics of OO design such as size, complexity, coupling and cohesion, and use a proportion to an average as the measurement indicator. It is easy to classify classes which have a result above the average, and to predict classes which reduced the quality of OO design. They will be modified to hold the average. Proposed metrics are analytically evaluated by Weyuker's nine properties. They are satisfied with seven properties except two properties do not apply to OO metrics. Also, we design a quality assessment system, ASSOD(ASsessment System of Object oriented Design), to measure the quality of the OO design independent of the platform.

† 정 회 원 : 숙명여자대학교 대학원 전산학과
†† 정 회 원 : 숙명여자대학교 정보과학부 교수
논문접수 : 1999년 11월 22일, 심사완료 : 1999년 12월 14일

1. 서 론

객체지향 방법을 이용한 소프트웨어의 개발을 관리하려면 객관적인 측정 기술이 필요하다. 장래 수행될 프로젝트에 반영될 여러 가지 중요한 자료가 되기 때문이다. 따라서 현재 객체지향 패러다임에서 소프트웨어의 품질 측정은 중요한 관심의 대상이 되고 있다[1].

객체지향 기술의 특징은 프로그램을 뚜렷하게 구별되는 단위인 객체(object)로 분할할 수 있다는 것이다. 구별된 단위들은 잘 정의된 인터페이스를 이용하여 상호 작용할 수 있고, 잘 분할된 시스템은 수정할 때 그 영향권이 적어지므로 변경 작업이 쉬워진다. 즉, 잘 설계된 객체는 시스템의 품질에 직접적으로 큰 영향을 주게 되고, 유지보수 비용을 줄일 수 있게 된다. 또한, 구현하는 동안 직면하는 설계 결점들을 초기에 발견할수록 개발비용이 적어지므로, 객체지향 설계의 품질을 측정하는 것이 매우 중요해졌다[2][3].

그러나, 객체지향 기술을 사용한 프로젝트는 전통적인 방법과는 많은 차이점을 가지고 있다. 먼저 재사용과 품질이 강조된 반복적 라이프사이클을 갖는다. 또한 소프트웨어 구조가 매우 다르다. 제어 흐름의 계층적 분해가 절차적 방식의 구조적 특징이라면, 객체지향 소프트웨어는 순수한 일대일 관계이다. 또한 추상화 수준이 시스템, 부 시스템(subsystem), 클래스, 메소드 등으로 잘 나누어져 있고, 상속 관계가 소프트웨어 구조를 결정하는 중요한 요소가 된다. 따라서 절차적 중심의 소프트웨어에 대한 메트릭을 그대로 사용한다는 것은 무리가 있다.

지금까지의 객체지향 메트릭스에 대한 연구는 품질의 일부 요소만을 다루고 있는 것이 대부분이며 객체지향 소프트웨어의 전체 품질 특성을 목표로 메트릭을 제시한 연구는 매우 적다[4]. 또한, 이들은 클래스 사이의 관계 정보만을 기반으로 제안되어 왔다. 이들은 객체간의 메시지 전송과 같은 상호작용 측면을 효과적으로 예측, 측정하지 못하고 있으며, 이로써 객체지향 설계의 특성을 충분히 반영하지 못하고 있다.

실제로 객체지향 설계에 대한 메트릭스는 설계 자체에 대한 측정이 아니라, 설계의 특성에 대한 측정을 의미한다. 객체지향 설계의 특성은 소프트웨어의 크기, 복잡도, 결합도, 그리고 응집도로 크게 구분된다[5].

이에 본 논문에서는 객체지향 설계의 특성, 즉 설계의 크기, 복잡도, 결합도 및 응집도의 측면을 고려하여 설계의 품질을 평가하기 위한 메트릭 집합을 제시한다. 제시된 설계 메트릭 집합은 개별적인 클래스나 설계 구성요소(component)에 대한 평가를 통하여 전체 설계에 대한 품질을 평가할 수 있다. 또한, 기존의 메트릭들이 비공식적으로 정의되어 이해하기 어렵고, 그 결과를 한눈에 알아볼 수 없었던 단점을 극복하기 위하여, 각 메트릭은 공식적으로 정의되었다. 계산된 결과는 평균값에 대한 비율(proportion)로 나타냄으로써, 평균값을 상회하는 클래스 및 설계 구성요소들로 인하여 설계 전체의 크기나 복잡도가 증가되고 있다는 것을 예측할 수 있도록 하였다.

본 논문에서 제안하고 있는 메트릭 집합은 Weyuker의 측정 원칙[6]에 따라 분석적으로 평가된다. 그리고, 분산 인트라넷 환경에서 새로운 메트릭 값을 자동적으로 계산할 수 있도록 지원하는 정량화 분석 도구인 ASSOD (ASsessment System of Object-oriented Design)를 설계하였다.

2. 관련 연구

객체지향 소프트웨어는 기존의 전통적인 방법을 사용하여 개발된 소프트웨어와는 기본적으로 다르다. 그러므로 객체지향 소프트웨어를 위한 메트릭스는 이들과 구분되는 객체지향의 특성에 맞추어져야 한다. 객체지향 소프트웨어의 기본적인 특징으로는 지역화(Localization), 캡슐화(Encapsulation), 정보 은닉(Information hiding), 상속성(Inheritance), 그리고 추상화(abstraction)가 있다[7].

지역화는 프로그램 내에서 정보가 집중되는 방법을 알려주는 특성이다. 기존의 소프트웨어에서 지역화는 기능성(functionality)에 기반을 두고 있다. 그러나 객체지향 소프트웨어에서는 자료와 처리(process)를 함께 캡슐화 시킨 클래스에 기반을 두고 있다. 전통적인 소프트웨어에서는 지역화의 방법으로 기능성(functionality)을 강조하기 때문에, 메트릭스는 내부구조 또는 기능의 복잡도, 예를 들면 모듈의 길이, 싸이클로메틱 복잡도 등에 초점을 맞추어 왔다. 반면에, 객체지향 소프트웨어는 클래스에 기반을 두고 있으므로 완전한 한 개체로서 클래스에 적용되는 메트릭스가 정의되어야만 한다.

객체지향 소프트웨어에서의 캡슐화는 속성과 행위를 포함하는 클래스에 의하여 정의되는 것으로서, 클래스의 상태(state)를 포함하게 된다. 그러므로, 메트릭스의 초점이 단일 모듈에서 자료와 처리 모듈의 패키지로

관점이 변하게 된다.

정보 은닉은 시스템 구성요소의 기능적으로 상세한 부분은 감추고, 그 구성요소에 접근하기 위하여 필요한 최소한의 정보만을 제공하는 것이다. 잘 설계된 객체지향 소프트웨어는 정보 은닉의 특성을 잘 나타낸 것이다. 그러므로, 은닉이 성취된 정도의 척도를 제공하는 메트릭스는 객체지향 설계의 품질에 대한 척도를 제공하게 되는 것이다.

상속성은 일반적으로 전통적인 소프트웨어에서는 지원하지 않는 특성이며, 계층 관계에 놓여 있는 클래스 또는 객체간에 속성이나 메소드의 상속이 이루어질 수 있음을 의미한다. 상속성은 객체지향 소프트웨어에서 주축이 되는 특성이기 때문에 많은 객체지향 메트릭스는 상속성에 초점을 맞추고 있다.

추상화는 구성요소에 대한 일반적인 관점을 제공하는 것이며, 객체지향 소프트웨어에서 추상화는 클래스로 표현된다. 그러므로, 객체지향 메트릭스는 클래스의 관점에서 추상화의 특성을 고려하게 된다.

기존의 소프트웨어 메트릭스에 관한 연구는 이러한 객체지향 소프트웨어의 특성을 고려하지 않는 주로 절차적 소프트웨어에 관한 것이었다. 그러므로, 이들 특성들을 반영한 객체지향 메트릭스에 대한 연구가 필요하다.

현재까지 소개되어진 객체지향 소프트웨어 품질 메트릭은 주로 객관적 내부 특성에 관한 것이다. 이것은 외부 특성을 예측하기 위한 기초 자료가 되는 객관적 메트릭 방안이다. 객체지향 메트릭에 대한 연구는 품질의 일부 요소만을 다루고 있는 것이 대부분이며, 설계의 전체 특성을 목표로 제시한 연구는 매우 적다. Chidamner와 Kemerer는 설계의 규모와 복잡도에 영향을 주는 요소를 파악하기 위하여 6가지 메트릭 집합을 제안하였다[8]. 이는 클래스당 가중치를 가진 메소드수, 상속 트리의 깊이, 자식 노드의 수, 객체간의 결합도, 클래스에 대한 반응도, 그리고 메소드에서의 응집도 결핍과 같은 클래스 기반 설계 메트릭이다. Dumke는 시스템 수준, 클래스 수준, 메소드 수준으로 분류하여 복잡도를 측정하기 위한 메트릭을 제시하고 있으며, Bieman과 Karunanithi는 수정 없이 그대로 재사용하는 경우와 수정하여 재사용하는 형태로 분류하여 재사용성을 나타내는 척도들을 제시하였다[1]. Loernz와 Kidd가 제안한 객체지향 소프트웨어 메트릭스[9], Li와 Henry의 객체지향 메트릭스[10] 등도 알려져 있다. 또한, Sharble과 Cohen은 객체지향 개발 방법중 두가지 스타일에 대한 비교를 위한 척도들을 제시하기도 하였다[11].

그러나 이들은 클래스 사이의 관계 정보만을 기반으로 제안되어 왔기 때문에, 객체간의 메시지 전송과 같은 상호작용 측면을 효과적으로 예측, 측정하지 못하고 있으며, 이로써 객체지향 설계의 특성을 충분히 반영하지 못하고 있다. 또한, 이들은 비공식적으로 정의되어 이해하기 어렵고, 그 결과를 한눈에 알아볼 수 없다. 뿐만 아니라, 단지 메트릭스의 성질이나 개념만을 언급하고 있으며, 그들을 측정하기 위한 구체적인 절차를 설명하지 못하고 있는 것이 대부분이다.

3. 새로운 객체지향 설계 메트릭스

3.1 객체지향 설계의 특성

실제로 객체지향 설계에 대한 메트릭스는 설계 자체에 대한 측정이 아니라, 설계의 특성에 대한 측정을 의미한다. 새롭게 제안된 설계 메트릭스는 이들 설계 특성을 고려하여 정의되었다. 본 논문에서 고려한 설계의 특성은 크게 크기(size), 복잡도(complexity), 결합도(coupling), 응집도(cohesion)이다. 각 소프트웨어에는 그 성격에 맞는 설계 특성을 가지고 있으나, 위의 4가지 특성은 모든 객체지향 소프트웨어에서 고려되어야 할 설계 특성으로 지적되고 있는 항목들이다[5].

3.1.1 소프트웨어의 크기

소프트웨어의 크기는 산출물의 내부적인 속성으로서, 대부분의 제품이나 공정 그리고 프로젝트의 측정(measures)에 기여하게 된다. 크기는 프로젝트에 대한 노력과 일정을 예측하는데 사용되는 것을 비롯하여, 많은 예측 시스템에서 사용되고 있다. 부품(component)이나 클래스 수준에서의 크기는 프로젝트를 정확하게 진행하는데 사용될 수 있고, 소프트웨어의 완성까지 관리에 대한 좀더 정확한 견적을 만들어 낸다. 또한, 주어진 프로젝트 상에서 재사용의 수준을 측정하는데 사용될 수도 있다.

3.1.2 복잡도

복잡도는 문제를 이해하고 해결하는 데 매우 중요할 뿐만 아니라, 소프트웨어를 이해하고 사용하고 수정하는 데에도 중요한 역할을 한다. 복잡도는 시스템이나

컴포넌트가 얼마나 이해하고 검증하기 어렵게 설계, 구현 되었는지의 정도를 나타낸다. 그러므로, 복잡도는 신뢰성에 부정적인 영향을 주고, 소프트웨어를 개발하는데 필요한 노력을 나타내는 척도가 된다고 알려져 왔다.

3.1.3 결합도

한 시스템의 요소들간의 연결의 성질(nature)과 정도(extent)를 나타내는 것이 결합도이다. 한 시스템의 요소들 사이에 논리적이거나 물리적인 연결의 형태는 결합도의 형태로 정의된다. 그러므로, 결합도는 설계의 구성요소(components) 사이의 물리적 연결의 강도에 대한 측정으로 정의할 수 있으며, 이것은 클래스 범주들 사이에서, 클래스들 사이에서, 또는 한 클래스의 요소들 사이에서 서로에게 얼마나 의존하고 있는지에 대한 측정이 된다. 결합도를 측정한다는 것은 시스템에 대한 변화의 범위를 예측하고 조절할 수 있다는 것을 의미한다.

3.1.4 응집도

응집도는 모듈의 구성요소 또는 클래스 상의 관계 또는 연결의 관점에서 정의된다. 응집된 구성요소들은 응집되지 않은 구성요소보다 재사용하기가 쉽다. 완전하고 응집된 구성요소는 그 구현에 대한 추상화를 요구하는 어떤 응용에서도 사용될 수 있다.

특히, 상속성은 객체지향 패러다임의 중요한 개념중의 하나이고, 그 사용이 장려된 것이다. 상속성을 결합도로 생각한다면, 높은 응집력과 낮은 결합도일수록 좋은 객체지향 설계라는 일반적인 생각에 위배된다[12]. 그러므로, 상속 관계의 클래스들은 서로 깊이 관련이 있는 한 단위로 생각할 수 있고 응집도로 측정될 수 있다.

3.2 새로운 설계 메트릭스의 제안

본 논문에서 제안하는 설계 메트릭스는 객체지향 설계의 크기, 복잡도, 응집도, 결합도 특성으로 나누어 측정하며, 각 특성은 여러 가지 속성을 표현하는 메트릭 집합으로 정의된다. 그러므로 설계의 품질은 정의 1과 같이 나타낼 수 있다.

정의 1. 객체지향 소프트웨어의 설계 품질 QoD(Quality of Design)는 은 4개의 인자를 갖는 함수로 표현한다.

$$QoD = f(SI, CMX, CPL, COH) \quad \text{(식 1)}$$

(식 1)에서, 4개의 인자는 설계의 특성을 표현하는 함수값으로서, 다음과 같이 정의된다.

정의 1.1 설계의 크기를 나타내는 인자 SI는 패키지 당 평균 클래스의 수(ACP), 패키지 당 평균 메소드의 수(AMEP), 그리고 패키지의 수(NPS)를 속성으로 갖는 함수이다.

$$\text{설계 } D\text{의 크기 } SI(D) = f_1(ACP, AMEP, NPS)$$

정의 1.2 설계의 복잡도를 나타내는 인자 CMX는 클래스 당 평균 공용 속성의 수(APAC), 클래스 당 평균 메소드의 수(AMC), 클래스 당 평균 메소드 호출의 수(ACMC), 상속 단위의 수(NOI), 그리고 다중상속을 받는 클래스의 수(NOMI)를 속성으로 갖는 함수이다.

$$\text{설계 } D\text{의 복잡도 } CMX(D) = f_2(APAC, AMC, ACMC, NOI, NOMI)$$

정의 1.3 설계의 결합도를 나타내는 인자 CPL은 클래스 당 평균 포함하고 있는 클래스의 수(ACO), 클래스 당 평균 메시지 패싱의 수(AMP), 그리고 클래스 당 평균 부 클래스의 수(ASU)를 속성으로 갖는 함수이다.

$$\text{설계 } D\text{의 결합도 } CPL(D) = f_3(ACO, AMP, ASU)$$

정의 1.4 설계의 응집도를 나타내는 인자 COH는 속성에 대한 메소드의 평균 참조 횟수(ARM), 상속 단위에서 평균 상속 깊이(AID), 상속 단위에서 부 클래스에 의하여 재 정의된 메소드의 평균 수(AOO), 그리고 상속 단위에서 부 클래스에 의하여 추가된 메소드의 평균 수(AOA)를 속성으로 갖는 함수이다.

$$\text{설계 } D\text{의 응집도 } COH(D) = f_4(ARM, AID, AOO, AOA)$$

3.2.1 크기(size)에 대한 메트릭 집합

설계의 크기 측면을 평가하기 위한 다음의 메트릭 집합을 제안한다. 이것은 소프트웨어의 크기와 프로그램 개발 노력에 대한 척도로서 사용될 수도 있다. 객체지향 방법에서는 유사한 성격을 가지고, 서로 연관 관계가 많은 클래스들을 묶어 하나의 패키지(package) 단위로 그룹화(grouping)한다

[1]. 그러므로, 설계의 크기를 측정하기 위하여 본 논문에서 제안하고 있는 메트릭들은 패키지 단위로 측정하여 전체 시스템의 규모를 쉽게 파악할 수 있도록 정의하였다.

① 패키지 당 평균 클래스의 수
(Average of Classes per Package, ACP)

정의 2. $ACP(S) = \frac{\sum_{i=1}^{n} NOC(P_i)}{n}$,

$NOC(P_i)$는 패키지 (P_i)에 있는 클래스의 수를 계산한다.

(그림 1)의 경우, $NOC(ATM\ Interface) = 7$, $NOC(Transaction\ Management) = 2$, $NOC(Account\ Management) = 3$이다. 그러므로, $ACP(ATM) = (7+2+3)/3 = 4$가 된다. 소프트웨어의 크기는 프로젝트에 대한 노력과 일정을 예측하는데 사용될 수 있는데, ACP값보다 높은 값을 갖는 패키지의 경우, 좀더 많은 개발 시간과 노력이 필요하다는 것을 직관적으로 알 수 있다.

② 패키지 당 평균 메소드의 수
(Average of MEthods per Package, AMEP)

정의 3. $AMEP(S) = \frac{\sum_{i=1}^{n} TM(P_i)}{n}$

$TM(P_i) = \sum_{k=1}^{m} NOM(C_k)$이고, $NOM(C_k)$는 클래스 C_k에 있는 메소드의 수를 계산한다. 메소드의 수를 측정함으로서 구현 단계의 프로그램 코딩 노력을 예측할 수 있다.

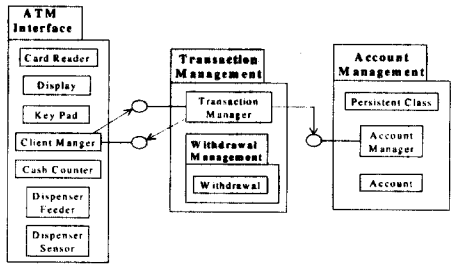
(그림 1)에서 ATM Interface, Transaction Management, 그리고 Account Management에 대한 TM 값이 각각 21, 7, 12라면, $AMEP(ATM)$은 13이 된다.

③ 패키지의 수
(Number of Packages in a System, NPS)

정의 4. $S = \{P_1, P_2, \cdots, P_n\}$라면, $NPS(S) = |S|$

P_i는 시스템 S의 임의의 패키지이고, $1 \le i \le n$ 이다.

$NPS(S)$는 시스템 S에 있는 패키지의 수이다. 소프트웨어 설계에서 중요한 문제중의 하나는 시스템을 변경하기 쉽도록 작은 단위로 나누는 것이다. 작은 단위의 패키지로 나눔으로써 시스템에 대한 이해를 쉽게 하고 구현 과정을 단순화시킬 수 있기 때문에, 시스템의 분할은 중요하다. (그림 1)에서의 $NPS(ATM) = 3$이다. 효율적인 클래스 설계 이론에 따르면 분할된 부 시스템의 수는 7±2정도가 바람직하다[4]. 따라서, (그림 1)의 ATM은 분할이 적절하게 이루어졌다고 할 수 있다.



(그림 1) 간단한 ATM의 패키지에 대한 예[13]

3.2.2 복잡도(Complexity)에 대한 메트릭 집합

각 클래스 자체의 복잡도를 측정하는 것이 전체 설계의 복잡도가 되는 것은 아니지만, 시스템이나 구성 요소가 이해하고 검증하기 어렵게 설계되었는지에 대한 척도가 될 수 있다.설계의 복잡도는 소프트웨어 개발 노력에 대한 척도가 될 수 있으며, 복잡한 설계는 유지보수를 어렵게 하고, 소프트웨어를 이해하기 어렵게 만든다.

본 논문에서 제안하고 있는 설계의 복잡도를 측정하기 위한 메트릭 집합은 다음과 같다.

① 클래스 당 평균 공용 속성의 수
(Average of the Public Attributes per Class, APAC)

정의 5. $APAC(S) = \frac{\sum_{i=1}^{n} NPA(C_i)}{n}$,

$NPA(C_i)$ = 클래스 C_i에 있는 공용 속성의 수를 계산한다.

(그림 2)는 라이브러리(library) 시스템에 대한 클래스 다이아그램이다[14]. 모든 클래스가 공용속성을 가지고 있지 않으므로, NPA 값은 모두 0이 되고, 따라서 $APAC(Library)$는 0가 된다. 클래스의 속성은 전용(private)을 원칙으로 하므로, 공용속성의 수가 많아 질수록 복잡도는 증가한다.

② 클래스 당 평균 메소드의 수
(Average of the Method per Class, AMC)

정의 6. $AMC(S) = \frac{\sum_{i=1}^{n} NOM(C_i)}{n}$,

$NOM(C_i)$ = 클래스 C_i에 있는 메소드의 수이다[3].

(그림 2)의 클래스 Item, Title, Magazine Title, Loan, Book Title, Reservation, 그리고 Borrower information에 대한 NOM 값은 각각 3, 1, 0, 0, 0, 1, 그리고 1이다.

그러므로, $AMC(Library) = (3+1+0+1+0+1)/7 = 0.86$이 된다. 클래스에 선언된 메소드의 수가 증가할수록 더 특정적이고 복잡한 클래스가 된다. AMC 값보다 높은 값을 가진 즉, 평균값보다 많은 수의 메소드를 가진 클래스에 대하여는 다시 검토할 필요가 있다는 것을 암시한다.

③ 클래스 당 평균 메소드 호출의 수
(Average of the Calling Method per Class, ACMC)

정의 7. $ACMC(S) = \frac{\sum_{i=1}^{n} NCM(C_i)}{n}$,

$NCM(C_i)$ = 클래스 C_i의 메소드 호출 수이다.

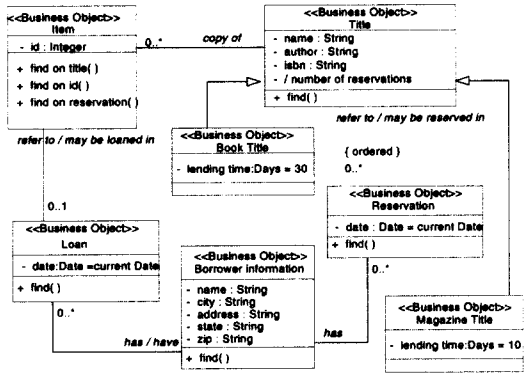
(그림 3)은 고객 관리 시스템의 시퀀스 다이아그램의 일부이다[15]. 이 경우, 클래스 frmCustomer, CCustomerCtrl, 그리고 CCustomer의 메소들 호출의 수는 각각 3, 3, 2이다. 그러므로, (그림 3)의 클래스 당 평균 메소드 호출의 수는 $(3+3+2)/4 = 2$가 된다.

④ 상속 단위의 수(Number of the Inheritance Unit, NOI)

정의 8. $NOI(S) = |R|$,

$R = \{C_i \mid C_i$는 상위 클래스는 없고, 부 클래스만 존재하는 클래스$\}$이다.

(그림 2)에서, R = { Title }이며, $NOI(Library)$는 1이다. 상속 단위의 수가 많아질수록, 시험하기 어려운 복잡한 설계가 된다.



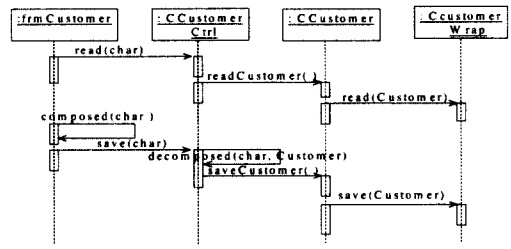
(그림 2) Library system의 클래스 다이아그램

⑤ 다중상속을 받는 클래스의 수
(Number of the Multiple Inheritance Class, NOMI)

정의 9. $NOMI(S) = |ML|$,

$ML = \{ C_i \mid C_i$는 부모 클래스의 수가 2개 이상인 클래스, $1 \le i \le n\}$이다.

다중상속을 받는 클래스는 재사용이 어렵고, 구현을 복잡하게 한다. 이런 클래스들이 많아지면 유지보수하기 어려운 복잡한 설계가 된다.



(그림 3) 고객 관리 시퀀스 다이아그램

3.2.3 결합도(Coupling)에 대한 메트릭 집합

결합도는 클래스 범주들 사이에서, 클래스들 사이에서, 또는 한 클래스의 요소들 사이에서 서로에게 얼마나 의존하고 있는지에 대한 측정이 된다. 클래스들 간의 과도한 결합은 설계의 모듈화를 해치고, 그 결과 유지보수와 재사용 등을 어렵게 한다. 모든 결합도를 제거할 수는 없으며, 모듈화를 향상시키고 캡슐화를 증진시키기 위해서 클래스들간의 결합은 제한되어야 한다. 다음은 클래스간의 결합도를 측정하는 메트릭 집합이다.

① 클래스 당 평균 포함하고 있는 클래스의 수
(Average of the Containing classes per Class, ACO)

정의 10. $ACO(S) = \frac{\sum_{i=1}^{n} CON(C_i)}{n}$,

$CON(C_i)$는 클래스 C_i가 포함하고 있는 클래스의 수, n은 $CON(C_i)$값이 1이상인 클래스의 갯수이다.

(그림 4)은 클래스 사이의 관계를 나타내고 있는 예제이다[14]. (그림 4)에서 각 클래스가 포함하고 있는 클래스의 수를 계산하면, $CON(Canvas) = 1$, $CON(Group) = 1$, 그리고 $CON(Polygon) = 1$이며, $ASO(S) = (1+1+1)/3 = 1$가 된다. 평균값을 넘는 클래스들은 결합도를 줄이는 방향으

로 재검토되어야 함을 의미한다.

② 클래스 당 평균 메시지 패싱의 수
(Average of the Message Passing per Class, AMP)

정의 11. $AMP(S) = \frac{\sum_{i=1}^{n} NOMP(C_i)}{n}$,

$NOMP(C_i)$는 클래스 C_i의 메시지 패싱의 수,
n은 $NOMP(C_i)$ 값이 1이상인 클래스의 수이다.

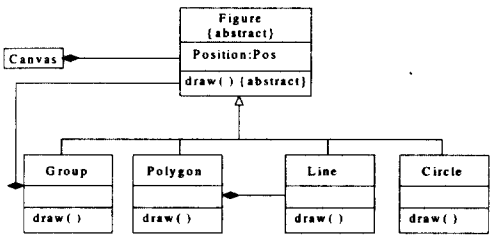
이것은 각 클래스가 얼마나 많은 연관관계를 가지고 있는 가에 대한 측정이다. (그림 2)에서, 클래스 Item, Title, Reservation, Loan, 그리고 Borrower information에 대한 메시지 패싱의 수는 각각 2이다. 그러므로, $NOMP(S) = (2+2+2+2+2)/5 = 2$이다. 클래스 당 평균 메시지 패싱의 수가 많아질수록 클래스간의 결합도가 높다는 것을 의미하며, 평균값을 넘는 클래스들은 결합도를 줄이도록 재설계되어야 한다.

③ 클래스 당 평균 부 클래스의 수
(Average of the Subclasses per Class, ASU)

정의 12. $ASU(S) = \frac{\sum_{i=1}^{n} NSU(C_i)}{n}$,

$NSU(C_i)$는 클래스 C_i의 부 클래스의 수,
n은 $NSU(C_i)$ 값이 1이상인 클래스의 갯수이다.

(그림 2)에서 $NSU(Title) = 2$이고, (그림 3)에서 $NSU(Figure) = 4$이다. 만약, 이들 두 클래스가 한 시스템 S에 존재하는 클래스라면 $ASU(S) = (4+2)/2 = 3$으로 계산된다. ASU 값이 높다는 것은 그만큼 상속이 많이 이루어지고 있다는 의미가 된다.



(그림 4) 간단한 클래스 다이아그램

3.2.4 응집도(Cohesion)에 대한 메트릭 집합

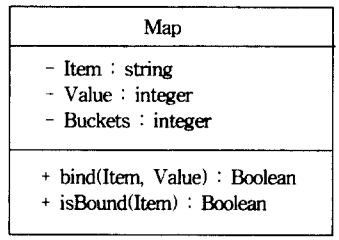
설계의 응집도를 측정하기 위한 새로운 메트릭 집합을 다음과 같이 제안한다. 클래스의 추상화 정도와 클래스 사이의 상속 관계에 대한 척도로 사용될 수 있다. 응집되어 설계된 구성 요소들은 재 사용하기 쉬우며, 사용하고 이해하기 쉬워진다. 속성에 대한 메소드의 평균 참조 횟수는 클래스의 응집도에 대한 메트릭이며, 나머지 3개의 메트릭들은 클래스 사이의 응집도를 평가하기 위한 것이다.

① 속성에 대한 메소드의 평균 참조 횟수
(Average of the Reference for Attributes by Methods, ARM)

정의 13. $ARM(S) = \frac{\sum_{k=1}^{n} |RS(A_k)|}{n}$,

$RS(A_k)$는 속성 A_k를 참조하는 메소드의 집합,
n은 속성의 갯수를 말한다.

(그림 5)에서 클래스 Map의 속성 Item을 참조하는 메소드는 bind(), isBound()이므로 RS(Item) = {bind(), isBound()}가 된다. 또한, 속성 Value를 참조하는 메소드는 bind()이므로 RS(Value) = {bind()}가 된다. 그러므로, 각 속성에 대한 RS값은 2와 1이 된다.



(그림 5) 클래스 "Map"

② 평균 상속 깊이(Average of Inheritance Depth, AID)

정의 14. $AID(S) = \frac{\sum_{i=1}^{n} DIT(IU_i)}{n}$,

IU는 상속 단위(Inheritance Unit)를 나타내며,
n은 상속 단위의 갯수이다.

DIT는 상속 단위의 깊이이고 [8]에서 정의되었다. (그림 6)은 상속 관계를 나타내는 클래스 다이아그램이다 [13]. 클래스 Bond의 조상 클래스 수는 1이고, 클래스 SmallCapStock의 조상 클래스 수는 2가 되며, 이 상속 단위의 깊이는 3이다. 상속 깊이가 증가할수록 상속되는 속성과 클래스가 많아지므로, 기능적으로 응집되어 있는

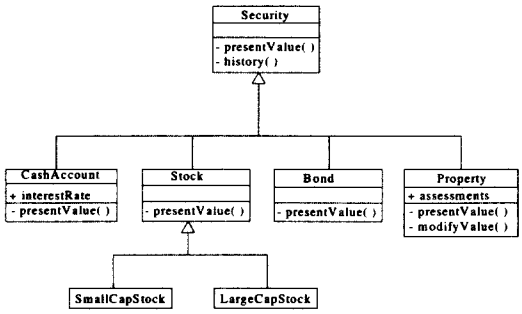
상속 단위를 이루게 된다고 할 수 있다. 그러나, 상속 깊이가 지나치게 커지면 복잡한 설계가 된다.

③ 상속 단위에서 부 클래스에 의하여 재 정의된 메소드의 평균 수
(Average of Operations Overridden by a subclass in an Inheritance Unit, AOO)

정의 15. $AOO(S) = \frac{\sum_{i=1}^{n} NOO(IU_i)}{n}$,

n은 상속 단위의 개수이며, IU는 상속 단위이다.

NOO는 [9]에서 정의한 부 클래스에 의하여 재 정의된 메소드의 수를 말한다. (그림 6)에서 클래스 Security, Stock, 그리고 SmallCapStock은 하나의 상속 단위를 이루며, 메소드 presentValue()가 재 정의되고 있으므로 NOO값은 1이 된다.



(그림 6) 상속 관계에 대한 예제

④ 상속 단위에서 부 클래스에 의하여 추가된 메소드의 평균 수
(Average of Operations Added by a subclass in an Inheritance Unit, AOA)

정의 16. $AOA(S) = \frac{\sum_{i=1}^{n} NOA(IU_i)}{n}$,

n은 상속 단위의 개수이며, IU는 상속 단위이다.

NOA는 [9]에서 정의한 부 클래스에 의하여 추가된 메소드의 수를 말한다. (그림 6)의 클래스 Property에 정의된 메소드 modifyValue( )는 상위 클래스에서 정의되지 않고 그 클래스에서 추가된 메소드이다. 추가된 메소드가 많아질수록 더욱더 특정적인 클래스가 된다.

4. 메트릭스의 평가

본 논문에서는 제안된 메트릭스를 평가하기 위하여 Weyuker의 성질을 이용한다[6]. 이 평가 원칙은 몇 가지 비평을 받고 있다. 그러나, 이들이 현재까지도 널리 알려져 있고 적용되고 있기 때문에, 이 평가 원칙을 사용하였다.

아래 표기법에서 P와 Q는 클래스를 나타내며, $\mu(P)$와 $\mu(Q)$는 각각 P와 Q의 메트릭을 나타낸다.

성질 1. $\mu(P) \neq \mu(Q)$인 P와 Q가 존재한다.

이것은 모든 클래스가 한 메트릭에 대해 같은 값을 가질 수 있는 것은 아니며, 그렇지 않으면, 측정으로서의 값을 상실할 수 있다는 것을 암시한다. 이 성질은 어떤 메트릭도 모든 객체지향 시스템에 대하여 같은 설계 특성을 갖지 않는다는 것을 반영하고 있으며, 위에서 제안한 메트릭스 모두가 만족하고 있다는 것을 알 수 있다.

성질 2. c를 음이 아닌 수라고 하면, 복잡도 c의 프로그램은 단지 유한히 많이 존재한다.

이것은 오로지 같은 메트릭 값을 갖는 유한개의 경우를 요구한다. 논문의 세계에서는 기껏해야 유한개의 응용을 다루고 있고, 각각의 응용은 유한개의 클래스를 갖기 때문에, 이 성질은 클래스 수준에서 측정된 어떤 메트릭에 의해서도 충족될 것이다. 그러므로, 본 논문에서 제안하고 있는 메트릭스 모두가 만족한다.

성질 3. $\mu(P) = \mu(Q)$인 별개의 클래스 P와 Q가 존재할 수 있다.

이것은 두 개의 클래스가 같은 메트릭 값을 가질 수 있다는 것을 의미한다. 즉, 두 개의 클래스가 같은 설계 특성을 가질 수 있다. 각 프로그램에 유일무이한(unique) 수치적 이름을 할당하고, 이 이름으로 그 프로그램의 복잡도를 다루는 메트릭은 모두 이 성질을 만족하므로[5], 역시 위에서 제안된 새로운 메트릭 집합도 성질 3을 만족하게 된다.

성질 1, 2, 그리고 3은 프로그램에 직접적으로 영향을 주지 않는 것이며, 메트릭으로서 만족되어져야 하는 가장 기본적인 성질들이다. 그러므로, 본 논문에서 제안하고 있는 객체지향 설계 특성에 대한 메트릭 집합은 소프트웨어 메트릭스로서 필수적인 성질들을 갖추고 있다고 말할 수 있다.

성질 4. 같은 기능을 수행하는 두 클래스 설계 P와 Q가 존재한다고 할 때, 이것이 $\mu(P) = \mu(Q)$를 의미하지 않는다.

이 성질이 내포하고 있는 것은 두 개의 클래스 설계가 같은 기능을 수행할 지라도, 구현에 대한 상세한 부분은 설계를 측정하는데 중요한 역할을 한다는 것이다. 이 성질은 성질 1과 같은 뜻을 가지고 있으며, 결국 본 논문의 메트릭스가 만족하고 있는 성질이다.

성질 5. 모든 클래스 P와 Q에 대하여 P;Q가 P와 Q의 연결(concatenation)을 의미할 때, $\mu(P) \leq \mu(P;Q)$, 그리고 $\mu(Q) \leq \mu(P;Q)$이다.

이것은 두 개의 클래스의 조합에 대한 메트릭은 구성요소인 클래스들 각각의 메트릭보다 결코 작아질 수 없다는 것을 의미한다. 본 논문의 메트릭스는 평균값에 대한 비율을 사용하고 있으므로, 클래스의 개수와 직접적으로 관련이 없는 NPS나 AID와 같은 메트릭만이 이 성질을 만족하지 않게 된다.

성질 6. $\exists P, \exists Q, \exists R$ 일 때, $\mu(P) = \mu(Q)$가 $\mu(P;R) = \mu(Q;R)$을 의미하지 않는다.

이것은 P와 R 사이의 상호작용이 Q와 R사이의 상호작용과는 다를 수 있다는 것을 의미하는 것으로서, 위에서 제안된 모든 메트릭스가 만족하고 있는 성질이다.

성질 7. P안에 있는 원소들이 Q의 원소들의 순서와는 다르다면, $\mu(P) \neq \mu(Q)$인 클래스들 P와 Q가 존재한다.

성질 7은 프로그램의 복잡도가 그 프로그램 내에서의 문장의 순서를 반영해서 문장들 사이의 잠재적인 상호작용을 이루도록 해야만 한다는 것을 의미한다. 그러나, 객체지향 설계에서 문장의 순서는 실제 실행이나 클래스, 메소드 그리고 속성의 사용과는 상관이 없기 때문에 본 논문에서 제안된 메트릭스는 성질 7을 만족하지 않는다고 할 수 있다.

성질 8. P가 Q의 새로운 이름이라면, $\mu(P) = \mu(Q)$이다.

이것은 측정된 개체의 이름이 바뀔 때, 메트릭은 바뀌지 않고 유지될 것을 요구한다. 이 논문에서 제안된 모든 메트릭이 클래스 수준에서 측정되고, 클래스나 메소드, 인스턴스 변수의 이름에 의존하지 않으므로 이 성질을 역시 만족하게 된다.

성질 9. $\mu(P) + \mu(Q) \leq \mu(P;Q)$인 P와 Q가 존재한다.

이 성질은 두 개의 클래스가 결합될 때, 클래스들 간의 상호작용은 메트릭 값을 증가시킬 수 있다는 것이다. 그러나, [8]에서는 기억장치 관리나 실행시간 에러 감지와 같은 일이 클래스의 수가 많아질 때 더 어려워지며, 성질 9는 객체지향 소프트웨어 설계 메트릭에 대한 필수적인 특징이 될 수 없다고 말하고 있다.

<표 1>은 Weyuker의 9가지 성질에 대해 본 논문에서 제안하고 있는 메트릭스를 평가한 결과이다. 성질 5와 같은 경우는 2가지 메트릭만이 만족하지 않고 있으며, 성질 7과 성질 9은 객체지향 설계 메트릭의 필수적인 성질이 될 수 없으므로 평가 결과는 만족하지 않게 됨을 알 수 있다.

〈표 1〉 Weyuker's properties에 대한 평가

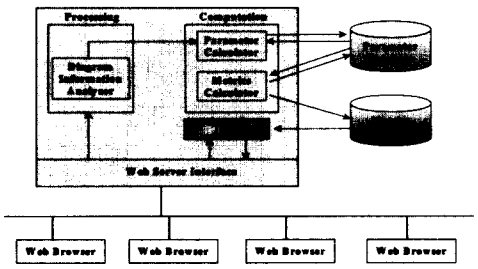
메트릭 \ 성질	1	2	3	4	5	6	7	8	9
ACP	○	○	○	○	○	○	×	○	×
AMEP	○	○	○	○	○	○	×	○	×
NPS	○	○	○	○	×	○	×	○	×
APAC	○	○	○	○	○	○	×	○	×
AMC	○	○	○	○	○	○	×	○	×
ACMC	○	○	○	○	○	○	×	○	×
NOI	○	○	○	○	○	○	×	○	×
NOMI	○	○	○	○	○	○	×	○	×
ACO	○	○	○	○	○	○	×	○	×
AMP	○	○	○	○	○	○	×	○	×
ASU	○	○	○	○	○	○	×	○	×
ARM	○	○	○	○	○	○	×	○	×
AID	○	○	○	○	×	○	×	○	×
AOO	○	○	○	○	○	○	×	○	×
AOA	○	○	○	○	○	○	×	○	×

5. 객체지향 설계 평가 시스템 ASSOD(ASsessment System of Object-oriented Design)의 설계

이 시스템은 분산 인트라넷 환경을 지원하기 위하여, 웹 브라우저 및 자바 애플릿으로 개발되어 플랫폼과 무관하게 사용할 수 있으며, SQL 문을 지원할 수 있는 함수 호출 인터페이스(CLI : Call Level Interface)의 일종인 JDBC를 이용하여 수집된 데이터와 계산된 메트릭 정보를 데이터베이스에 저장할 수 있도록 설계하였다.

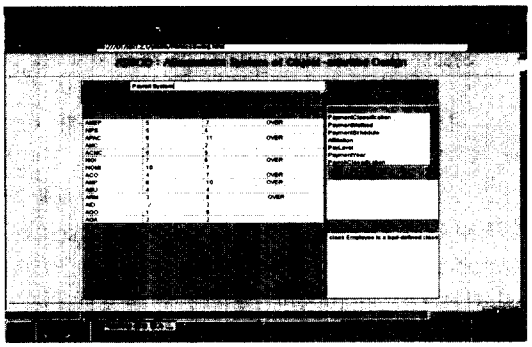
(그림 7)은 본 논문에서 제안하고 있는 메트릭 집합의 계산을 자동적으로 지원할 수 있는 분석 도구인 ASSOD의 구성도이다. 설계 단계에서 작성된 다이아그램을 분석하여 메트릭 계산에 필요한 정보를 추출하

는 과정은 다이아그램 정보 분석기(Diagram Information Analyzer)를 통하여 이루어진다. 파라메터 계산기(Parameter Calculator)는 설계의 크기, 복잡도, 결합도 및 응집도를 계산하는데 필요한 데이터를 분석해서 추출한다. 이렇게 추출된 데이터의 저장소는 파라메터 데이타베이스이다. 메트릭 계산기(Metric Calculator)는 각 설계 특성에 대한 메트릭 값을 계산하게 된다. 메트릭 데이터베이스는 계산된 메트릭 값과 사용자의 요구에 따라 설계의 품질 평가 결과를 보여줄 수 있도록 보고서를 작성하기 위하여 필요한 정보의 저장소이다.



(그림 7) 메트릭 지원 도구 ASSOD의 구성도

다음의 (그림 8)은 ASSOD의 사용자 인터페이스이다. 시스템 설계자는 웹브라우저와 데이터베이스 질의어를 통하여 설계에 대한 품질을 평가하게 된다. 설계자는 각 평가 항목에 대하여 평균값을 상회하는 클래스들을 쉽게 파악할 수 있다.



(그림 8) ASSOD의 사용자 인터페이스

본 논문에서 설계한 설계 평가 시스템 ASSOD는 분산 환경에서 평가하고자 하는 설계 산출물에 대한 품질을 측정하기 위하여 이해하기 쉽고 일관성 있는 인터페이스를 통하여 직접 접근할 수 있다. 특히 웹 브라우저 및 자바 애플릿으로 개발되어 플랫폼과 무관하게 사용할 수 있다는 장점이 있다. 또한, JDBC를 이용한 3 계층(3-tier)형태의 클라이언트/서버 구조를 이용하도록 설계하였다. 3-tier는 자바 애플릿이 직접 DBMS를 접근하는 것이 아니라, 중간에 있는 미들웨어를 거쳐 데이터베이스에 접근하게 된다. 이런 형태는 보안이나 확장성에서 2-tier보다 뛰어나다. 2-tier는 자바 애플릿에서 DBMS를 직접 접근하기 때문에, 인터넷을 사용하는 사람들 모두에게 노출된 상태이고 데이터베이스의 구조가 변경되면 데이터베이스를 직접 접근하는 모든 클라이언트들의 코드가 변경되어야 한다. 이것은 클라이언트의 수가 많은 경우에 상당히 많은 수정 작업을 필요로 한다.

그러나, 3-tier를 사용하는 경우에는 클라이언트가 직접 접근하는 것이 아니므로 클라이언트는 변경할 필요없이 미들웨어만 변경해 주면 된다. 그리고, 미들웨어 부분에서는 한 곳의 DBMS 서버에 많은 부하가 걸리면 다른 서버에서 서비스를 해주도록 로드 밸런싱이나, 많이 사용되는 데이터는 미들웨어가 저장하고 있다가 서비스해주는 캐싱 기능등을 제공할 수 있다.

〈표 2〉 물품보관 시스템에 대한 메트릭 결과

특성	메트릭	결과 값	평균을 넘는 요소들
크기	ACP	7.5	OrderPkg(10)
	AMEP	14.3	OrderPkg(26), CustomerPkg(16)
	NPS	6	
복잡도	APAC	0	
	AMC	2.5	COrder(9) 외 6개의 클래스
	ACMC	1.7	COrder(5) 외 3개의 클래스
	NOI	6	
	NOMI	0	
결합도	ACO	1.2	COrder(3)외 2개의 클래스
	AMP	3	COrder(4)외 5개의 클래스
	ASU	3	CCustomer(4)외 2개의 클래스
응집도	ARM	3.5	CItem(4)외 3개의 클래스
	AID	2.5	
	AOO	2	
	AOA	1	

6. 적용 사례 및 분석

6.1 품질 평가 사례

본 논문에서 제안한 객체지향 설계 메트릭스의 효율성을 판단하기 위하여 물품보관 시스템에 적용하여 분석하였다. 물품보관 시스템은 고객의 물품 보관에 대한 주문을 받아 특정 지역의 창고에서 일정 기간 동안 맡겨진 물품을 보관하는 서비스를 제공한다[15]. 이 시스템은 UML을 사용하여 모델링 하였고, 그 결과 작성된 다이아그램에 대하여 각 메트릭 값을 계산하였다.

<표 2>는 6개의 패키지로 분할된 물품보관 시스템에 대하여 크기와 복잡도에 관한 메트릭의 값을 계산한 결과를 보인 것이다. 각 메트릭의 결과는 평균값을 나타내므로 모든 클래스들에 대하여 계산된 값과 쉽게 비교해 볼 수 있다. 평균값을 상회하는 클래스나 구성요소들로 인하여 설계의 크기와 복잡도가 증가하게 된다는 것을 직관적으로 알 수 있으며, 쉽게 찾아낼 수 있었다.

6.2 비교 분석

품질 평가 메트릭스는 크게 두가지 측면에서 검토해 볼 수 있다. 먼저 메트릭의 다양성의 측면이다. 현재 많이 사용되고 있는 C&K 메트릭 집합은 전체 설계 특성보다는 설계의 규모와 복잡도와 같은 일부 요소만을 평가하고 있다. 그러나, 본 논문에서 제안하고 있는 메트릭스는 설계의 규모와 복잡도, 결합도, 응집도의 측면을 고려하고 있다.

또 다른 비교 관점은 품질 측정의 대상이다. 기존의 C&K 메트릭 집합은 설계 메트릭으로 제안되었으나, 그 평가 대상은 의사코드(pseudo code)이며 일부 메트릭은 원시코드를 가지고 측정할 수 있도록 제안되어 있다. 그러나 본 논문에서 제안하는 메트릭스는 소프트웨어 생명주기의 초기인 모델링 단계의 산출물을 가지고 평가할 수 있다. 이는 구현하는 동안 직면하는 설계 결점들을 개발 초기에 발견하여, 시스템의 품질 및 개발 비용에 직접적으로 영향을 주는 부분들을 효율적으로 재 설계 할 수 있다는 장점이 있다.

7. 결론 및 향후 연구과제

본 논문에서는 기존의 객체지향 메트릭스들이 고려하지 못했던 설계의 특성을 반영하는 메트릭 집합을 제시한다. 기존의 메트릭들이 비공식적으로 정의되어 이해하기 어렵고, 그 결과를 한눈에 알아볼 수 없었던 단점을 극복하기 위하여, 각 메트릭은 형식적으로 정의되하였다. 계산 결과는 평균값에 대한 비율(proportion)을 사용함으로써, 평균값을 상회하는 클래스 및 설계 구성요소들로 인하여 설계 전체의 품질에 대한 만족정도가 낮아지게 된다는 것을 예측할 수 있다.

본 논문에서는 객체지향 설계의 특성에 따른 메트릭 집합을 정의하고, 부분적인 예제를 통하여 각 메트릭 값을 계산하기 위한 방법을 설명하였다. 그리고 이들 메트릭 집합은 메트릭의 측정 원칙에 따라 분석적으로 평가되었으며, 그 결과로서 제안된 메트릭 집합이 메트릭에 대하여 요구되는 성질의 대부분을 만족하고 있음을 보았다.

객체지향 설계에 대한 평가 시스템 ASSOD는 플랫폼과 무관하게 사용할 수 있도록 웹 브라우저 및 자바 애플릿으로 설계되었으며 현재 개발중이다. 모든 설계자들이 분산 환경에서 이해하기 쉬운 인터페이스를 사용하므로, 보다 효과적으로 품질 평가가 이루어질 수 있을 것이다.

제안된 객체지향 설계 메트릭스는 개발 초기에 설계의 품질을 저하시키는 클래스나 구성요소들을 찾아내어 재 설계하기 위한 지침으로 사용될 수 있을 것이다. 예를 들어, 복잡도를 측정할 경우 평균값을 크게 웃도는 클래스는 다른 클래스들에 비해 설계를 복잡하게 하고, 코딩과 시험 등에 많은 노력을 기울여야 한다는 것을 쉽게 판단할 수 있게 된다. 이런 클래스들은 평균값을 기준 삼아 다시 설계되어야 할 것이다.

다만 더 많은 사례 연구와 품질 요소들과의 관계에 대한 연구가 필요하다.

참 고 문 헌

[1] 최은만, '소프트웨어 공학', 사이텍미디어, 1999.

[2] 김유경, 고병선, 고현희, 박재년, "객체지향 소프트웨어의 품질 메트릭스 및 품질 평가 도구의 설계에 관한 연구", 정보처리학회 학술발표 논문집, 제5권 제2호, 1998.

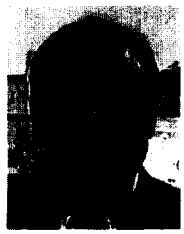
[3] 김유경, 고병선, 고현희, 박재년, "객체지향 소프트웨어의 품질 메트릭스에 관한 연구", 정보과학회 학술발표 논문집, 제26권 제1호, 1999.

[4] 최은만, 남윤석, "재사용 소프트웨어 품질 평가 도구 개

발", 정보처리학회 논문지, 제4권, 제8호, pp.1948-1960, 1997.
[5] Scott A. Whitmire, 'Object-Oriented Design measurement,' Wiley Computer Publishing, 1997.
[6] Weyuker E. J., "Evaluating software complexity measures," IEEE Transactions on Software Engineering, Vol.14, No.9, pp.1357-1365, 1988.
[7] Roger S. Pressman, 'Software Engineering, A Practitioner's Approach,' 4th Ed., McGraw-Hill, 1998.
[8] Shyam R. Chidamber, Chris F. Kemerer, "A Metrics suite for Object-Oriented Design," IEEE Transactions on Software Engineering, Vol.20, No.6, pp.476-493, June, 1994.
[9] M. Lorenz, J. Kidd, 'Object-Oriented Software Metrics : A Practical Guide,' Prentice-Hall, 1994.
[10] Li W., Henry S., "Object-oriented metric that predict maintainability," The Journal of Systems and Software, Vol.23, pp.111-122, 1993.
[11] Cohen S. S., Sharble R. C., "The object oriented brewery : comparison of two object-orientied development methods," ACM SIGSOFT Software Engineering Notes, Vol.18 No.2, pp.60-73, 1993.
[12] 박성희, 홍의석, 우치수, 김태균, "객체지향 프로그램에서 응집도, 결합도 측정 메트릭 집합", 정보과학회 논문지, 제25권 제12호, pp.1779-1787, 1998.
[13] Grady Booch, James Rumbaugh, Ivar Jacobson, 'The Unified Software Development Process,' Addison-Wesley, 1999.
[14] H. E. Eriksson, M. Penker, 'UML Toolkit,' Wiley Computer Publishing, 1998.
[15] 박현철, '객체지향 분석 설계 Visual C++ 프로그래밍', 비앤씨, 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