객체지향 분석 모델의 클래스 품질 척도

김유경[U] 박재년
숙명여자대학교 컴퓨터과학과
{ykkim, jnpark}@cs.sookmyung.ac.kr

Quality Metric for Classes in Object-Oriented Analysis Models

Yu-Kyung Kim[U] Jie-Nyun Park
Dept. of Computer Science, Sookmyung Women's University

요 약

객체지향 방법론은 캡슐화(encapsulation), 상속(inherit), 다형성(polymorphism)과 같은 개념을 이용하기 때문에 기존의 절차적 방법론과는 다른 척도가 필요하다. 본 논문에서 제안하는 척도는 객체지향 시스템의 개발 절차 가운데 분석 단계에서 추출할 수 있는 정보만을 사용하여, 클래스가 객체지향 개념에 따라 잘 구성되었는지를 측량하게 된다. 이를 위하여 본 논문에서는 클래스의 품질을 측량하기 위한 척도로 협력의 복잡도와 인터페이스 복잡도를 제안한다. 협력의 복잡도는 클래스가 잠재적으로 얼마나 복잡할 수 있는지를 측량하기 위한 것으로서 클래스가 가지는 책임의 개수를 조사하여 정의된다. 인터페이스 복잡도는 클래스와 협력 관계에 있는 다른 클래스들의 인터페이스를 조사하여 정의된다. 제안된 척도는 기존의 척도들이 가지고 있는 문제점을 해결하기 위하여 이해하기 쉬운 수학적 명세를 사용하였으며, 제안된 척도에 대한 수학적 증명과 사례 연구를 통한 검증을 하였다.

1. 서론

측량(Measurements)은 소프트웨어 개발 주기의 각 단계에서 산출물(product)과 개발 절차(process)를 제어하고 평가하기 위하여 필요하다[1]. 또한, 소프트웨어 측량은 개발비용이나 오류 발생률 등에 대한 예측에 사용된다. 프로젝트가 완성된 후에 비용을 예측하는 것은 쓸모 없는 일이 되며, 가능한 한 개발 초기에 계산 가능한 척도가 요구되고 있다.

또한, 객체지향 방법론이 캡슐화(encapsulation), 상속성, 다형성(polymorphism)과 같은 개념을 이용하기 때문에 기존의 절차적 방법론과는 다른 척도가 필요하다. 그러나, 객체지향 방법론에 대한 연구가 활발하게 진행되어 온 것과는 달리 객체지향 소프트웨어와 그 개발 절차의 품질을 향상시키기 위한 측량 방법은 널리 사용되지 못하였다[2].

본 논문에서는 객체지향 시스템의 개발 절차 가운데 분석 단계에서 추출할 수 있는 정보만을 사용하여, 클래스가 객체지향 개념에 따라 잘 구성되었는지를 측량할 수 있는 척도를 제안한다. 제안된 척도는 클래스가 잠재적으로 얼마나 복잡할 수 있는지를 측량하기 위한 협력의 복잡도와 클래스가 가지는 인터페이스가 얼마나 복잡한지를 측량하기 위한 복잡도로 구성된다.

이 척도들을 사용하여 시스템을 구성하는 클래스들을 개발하기 위한 노력을 예측할 수 있게 된다. 또한, 클래스 분할이나 결합과 같은 클래스 구성 과정에서 그 기준으로 사용할 수도 있다.

본 논문의 구성은 다음과 같다. 2장에서는 기존의 객체지향 소프트웨어에 대한 척도와 관련된 연구에 대하여 살펴보고, 3장에서는 본 논문에서 제안하고 있는 클래스 품질 척도에 대하여 설명한다. 4장에서는 제안된 척도에 대한 수학적인 증명과 함께 사례연구의 결과를 분석한다. 마지막으로 5장에서 결론 및 향후 연구과제를 기술한다.

2. 관련연구

현재까지 소개되어진 객체지향 분석 및 설계에 대한 품질 척도는 Chidamber와 Kemerer가 제안한 6가지 메트릭 집합[3], Lorenz와 Kidd의 객체지향 소프트웨어 메트릭스[4], 그리고 Li와 Henry의 객체지향 메트릭스[5] 등이 있다. 그러나, [3]에서 제안된 것과 같이 메소드의 매개변수나 메소드의 복잡도를 속성으로 정의된 척도들은 분석 초기에 제공되는 클래스 정보보다 훨씬 더 상세한 정보를 필요로 하기 때문에 분석 단계에서는 적용하기가 어렵다[6]. 객체지향 분석은 실세계의 시스템을 소프트웨어로 모델링하여 설명하는 단계로서, 문제 영역의 개념을 객체지향 모델 영역의 개념인 클래스로 추상화시키는 과정이다. 이러한 분석 활동의 결과로서 작성되는 분석 모델은 전체 시스템을 구성하는 클래스와 클래스가 수행해야 할 책임 및 그 책임을 수행하기 위하여 필요한 클래스간의 협력과 상속 관계를 포함한다. 따라서, 분석 모델에서 표현하고 있는 정보보다 훨씬 더 자세한 정보를 요구하는 척도들은 실제로 분석단계에서 사용할 수 없다. 예를 들어, RFC와 같은 경우는 클래스가 초기화되어 송신하는 각 메시지에 대한 정보를 요구하고 있다. 이것은 아마도 구현이 끝날 때 까지는 계산할 수

없을 것이다. 그러므로, 실제로 객체지향 분석과 설계를 위한 척도라고 말할 수 없다. 이로 인하여 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[7]. 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, 증명을 통한 검증과정을 포함하고 있다.

3. 분석 클래스에 대한 품질 척도

3.1 협력의 복잡도(Collaboration Complexity)

한 클래스가 다른 클래스와 연관(association)관계 또는 구성(aggregation)관계를 갖게 된다면, 이 관계를 협력이라고 정의한다. 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, r개 책임을 갖고, c개 협력자를 갖는 한 클래스 E에 대한 협력의 복잡도를 CC라고 표현한다면, 다음의 [정의 1]과 같다.

[정의 1] $CC(E) = r(1+c)$

여기에서 협력자의 수가 (1+c)가 된 것은 클래스는 자기자신과도 협력하기 때문이다.

또한, 한 클래스에 대한 협력의 복잡도는 완전한 한 시스템을 표현하는 클래스들의 집합에 대한 전체 복잡도를 계산하도록 확장할 수 있다. 클래스의 개수가 많아질수록 전체 시스템의 복잡도는 당연히 증가하게 된다. 따라서, 클래스의 구현에 따르는 비용과 노력을 비교하기 위한 의미 있는 값으로 평균적인 협력의 복잡도를 생각해 볼 수 있다. 그러므로, 시스템 S가 n개의 클래스들 $E_1, E_2, \cdots, E_n$로 이루어져 있다면, S의 평균 협력의 복잡도는 [정의 2]와 같다.

[정의 2] $CC(S) = \frac{1}{n}\sum_{i=1}^{n} CC(E_i)$

3.2 인터페이스 복잡도(Interface Complexity)

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

각 협력자에 대한 인터페이스 복잡도는 상속받은 책임의 수와 상속받지 않은 책임의 수를 더한 전체 책임의 개수로서 계산된다. 그러나, 책임의 개수가 증가할수록 인터페이스 복잡도는 지수 m에 의해 증가하게 된다는 가설[6]을 바탕으로 다음과 같이 정의할 수 있다. 계산의 편의를 위하여 m을 2로 선택한다면, 임의의 한 클래스의 전체 책임의 개수를 R이라고 할 때 인터페이스 복잡도는 R^2이 된다.

클래스 E가 전체 R개의 책임을 가지고 있고, c개의 다른 클래스들 $E_1, E_2, \cdots, E_c$ 과 협력하고 있다면, 인터페이스 복잡도를 IC라고 표현할 때, 인터페이스 복잡도 IC는 [정의 3]과 같이 주어진다.

[정의 3] $IC(E) = R^2 + \sum_{i=1}^{c} IC(E_i)$

여기에서, $IC(E_i)$는 클래스 E_i의 인터페이스 복잡도를 나타낸다.

또한, 한 클래스가 갖는 인터페이스 복잡도는 역시 완전한 한 시스템에 대한 전체 인터페이스 복잡도를 표현하도록 확장할 수 있다. 그러므로, n개의 클래스로 구성된 시스템 S의 평균 인터페이스 복잡도는 아래의 [정의 4]와 같다.

[정의 4] $IC(S) = \frac{1}{n}\sum_{i=1}^{n} IC(E_i)$

4. 클래스 품질 척도에 대한 증명

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

4.1클래스 분할

책임의 집합 P와 협력자의 집합 C를 가지는 클래스 E를 n개의 책임 $P_1, P_2, \cdots, P_n$과 $E_1, E_2, \cdots, E_n$인 n개의 클래스로 분할한다고 하자.

응집력이 약한 클래스는 보통 좀더 작은 응집된 클래스들로 분할된다. 이러한 경우에, 복잡도 척도가 분할된 클래스들에 대하여 좀더 낮은 복잡도를 제공한다는 것을 증명할 수 있다.

클래스 E가 r개 책임과 c개 협력자를 가지며, 상속받지 않으므로, $r = R$이다. 클래스 E가 각각 r_1과 r_2의 책임을 갖고, c_1과 c_2의 협력자를 갖는 좀더 작은 클래스들 E_1과 E_2로 분할되었다고 하자.

CC와 IC에 대하여 다음 식이 성립하게 된다.

$$\begin{aligned} CC(E) &= (c+1)r = (c_1+c_2+1)(r_1+r_2) \\ &= (c_1+1)r_1+(c_2+1)r_2+c_2r_1+c_1r_2 \\ &> CC(E_1)+CC(E_2) \end{aligned}$$

따라서, CC의 경우 클래스가 두 클래스로 분할 될 경우 좀더 낮은 복잡도를 갖게 된다.

만약, $\{F_1, \cdots, F_c\}$가 E의 협력자의 집합이고, 분할된 두 클래스 E_1과 E_2가 집합 $\{F_1, \cdots, F_{c_1}\}$과 $\{F_{c_1+1}, \cdots, F_c\}$의 협력자 집합을 갖는다면, 다음과 같이 IC를 계산할 수 있다.

$$\begin{aligned} IC(E) &= r^2 + \sum_{i=1}^{c} IC(F_i) \\ &= IC(E_1) + IC(E_2) + 2r_1r_2 \\ &> IC(E_1) + IC(E_2) \end{aligned}$$

따라서, IC의 경우에도 역시 클래스가 두 클래스로 분할 될 경우 좀더 낮은 복잡도를 갖게 된다.

4.2 클래스 결합

같은 협력자 집합을 가지고 있는 공통성이 많은 두 클래스들을 한 클래스로 결합할 때, 복잡도에 어떤 영향을 주게 되는지를 보이고자 한다. r_1과 r_2의 책임을 가지며, 공통 협력자의 집합 $C=\{F_1, F_2, \cdots, F_c\}$를 가지고 있는 두 클래스 E_1과 E_2를 생각해보자. 인터페이스의 복잡도는 모든 클래스들에 대해 상수 값을 갖으며, δ로 표시한다.

$$CC(E) = r(c+1) = (r_1+r_2)(c+1)$$
$$= CC(E_1)+CC(E_2)$$

원래의 경우와 결합된 경우에 대하여 CC의 값은 같다. 그러므로, 두 클래스가 결합된 경우의 복잡도는 인터페이스의 복잡도에 의해서만 판단될 수 있다.

$$IC(E) = r^2 + \delta = (r_1+r_2)^2 + \delta$$
$$IC(E_1) + IC(E_2) = {r_1}^2 + {r_2}^2 + 2\delta$$

$\delta > 2r_1r_2$ 라면, $IC(E) < IC(E_1) + IC(E_2)$이므로 두 클래스가 결합된 경우의 복잡도가 더 낮다는 것을 알 수 있다.

4.3 상속성

상속 메카니즘은 객체지향 시스템을 변화에 민감하지 않고, 따라서 좀더 유지보수하기 쉽게 만드는 것으로서 복잡도를 감소시키는데 매우 중요한 것이다. 척도에 대한 상속성의 영향을 시험해보기 위하여 상속받기 전과 후의 값을 비교해 보았다.

클래스 F와 G는 각각 책임 r_F와 r_G를 가지며, c_F와 c_G의 협력자를 가지고 있다. 이들 클래스의 공통의 책임을 r이라고 하면, 베이스(base) 클래스 E로부터 상속을 받고 있는 상속계층구조를 이룰 수 있다. 클래스 E는 클래스 F와 G에 정의된 책임들 중 일부로 구성된 책임들을 수행하게 된다. E가 r개 책임을 가지면, $r_F = r_{F'}+r$이고 $r_G = r_{G'}+r$ 이 된다. $r < r_F$이고 $r < r_G$이라고 하고, E는 어떤 구현부분도 가지고 있지 않고, 협력자의 수가 0인 추상화 클래스라고 가정하자.

$$CC_{F,G} = CC(F) + CC(G)$$
$$= r_F(1+|c_F|) + r_G(1+|c_G|)$$
$$CC_{E,F',G'} = CC(E) + CC(F') + CC(G')$$
$$= r(1+0) + r_{F'}(1+|c_F|) + r_{G'}(1+|c_G|)$$

따라서, $CC_{F,G} > CC_{E,F',G'}$가 되므로 상속 된 경우의 클래스의 협력의 복잡도는 감소하게 된다는 것을 알 수 있다.

그러나, 인터페이스 복잡도는 상속을 위한 베이스 클래스가 새로 생성이 되므로 생성된 클래스로 인하여 복잡도가 증가하게 된다.

그러나, 이 경우 이들 클래스들과 협력을 이루는 협력자들의 복잡도를 고려해 보면, 상속으로 인하여 협력자들의 인터페이스 복잡도는 감소하게 된다는 것을 알 수 있다. 따라서, 상속으로 인해 전체적인 시스템의 인터페이스 복잡도는 감소하게 된다고 할 수 있다.

4.4 사례 연구

[8]에서 한 시스템에 대한 여러 가지 가능한 설계를 제공하고 있다. 이에 대하여 CC와 IC를 계산하고, 그 결과에서 각 시스템에 대한 평균값과 전문가의 판정을 비교하였다. 그 결과, 9가지 비교 항목에 대하여 7가지 경우에 전문가의 선택과 일치하였다. 따라서, 제안된 척도는 비교 연구에서 77.8% 이내로 전문가의 선호도를 예측하였다. 전문가의 판정과 일치하지 않은 문제 영역은 협력과 인터페이스의 복잡도를 통해 상속으로 구현할 경우 복잡도를 줄일 수 있다는 것을 보이고 있으나, 문제 영역 자체가 단순하기 때문에 전문가들은 새로운 클래스를 생성하는 경우 인지적인 노력이 더 요구되는 것에 대한 부담감을 느끼는 것으로 나타났다.

또 다른 문제 영역에서 일치하지 않은 경우는, 단순한 클래스를 가진 설계와 좀더 응집되어 있는 책임을 가진 클래스로 분할되어 있는 설계를 비교한 경우이다. 좀 더 응집되어 있는 책임을 가진 경우 협력과 인터페이스의 복잡도가 낮아진다는 것을 보여주고 있다. 그러나 전문가들은 책임의 수가 적은 클래스의 경우 역시 새로운 클래스를 생성할 때의 부가적인 노력을 더 생각하는 것 같다.

5. 결론 및 향후 연구과제

본 논문에서는 객체지향 분석 모델을 이용하여 클래스의 복잡도를 측량하는 척도를 정의하였다. 기존에 연구되었던 척도들이 분석 단계에 나타나는 것보다 훨씬 더 상세한 정보를 요구하였기 때문에, 실제로 객체지향 분석 척도라고 말할 수 없다. 이러한 문제점을 해결하기 위하여, 분석 단계 정보만을 이용한 척도를 제안하였으며, 척도에 대한 수학적 증명과 사례 연구를 통하여 클래스 설계에 대한 선호를 예측할 수 있는 능력이 있음을 증명하였다.

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

그러나, 제안된 척도를 실제 개발 사례에 적용하여 충분한 사례 연구가 이루어져야 하며, 이를 통한 유효성 검증이 필요하다. 또한, 척도에 필요한 자료를 수집하고 복잡도를 계산하는 자동화 도구가 개발되어야 할 것이다.

참고 문헌

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

[2] R. S. Pressman, Software Engineering, 4th ed., McGraw Hill Comp., 1998.

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

[4] M. Lorenz and J. Kidd, Object-Oriented Software Metrics, Prentice-Hall, 1994.

[5] Li, Wei and S. Henry, "Maintenance Metrics for the Object-Oriented Paradigm", Proceedings of 1st International Software Metrics Symposium, pp.52-60, 1993.

[6] 김유경, 박재년, "클래스 기반 분석 모델에 대한 복잡도 메트릭", 제27회 한국 정보과학회 춘계학술발표대회 논문집, 제27권, 제 1호, pp.516-518, 2000.

[7] 김은미 외, "C++ 프로그램의 복잡도 척도 및 평가 도구", 정보과학회 논문지(C), 제 3권, 제6호, pp.656-665, 1997.

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