클래스 기반 분석모델에 대한 복잡도 메트릭

김유경° 박재년
숙명여자대학교 컴퓨터과학과
{ykkim, jnpark}@cs.sookmyung.ac.kr

A Complexity Metric for Class-Driven Analysis Models

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

요 약

객체지향 프로그램의 효율성과 설계의 품질을 평가하기 위해서는 필수적으로 정량적 메커니즘을 사용해야 한다.특히, 개발비용을 예측하기 위한 요구가 커질수록 소프트웨어 개발 생명주기의 초기에 측량해야 하는 필요성은 더욱 강해진다. 따라서, 분석단계 산출물에 대하여 이루어지는 측량은 나머지 개발 주기 단계에서 발견될 수 있는 결함에 대한 예측이나, 시스템의 개발에 소요될 노력과 비용을 예측하는데 사용될 수 있으므로 그 중요성이 더욱 강조되고 있다.

본 논문에서는 클래스 기반(class-driven)의 분석모델링 절차에 따라 작성되는 모델을 사용하여, 클래스의 책임과 협력 관계에 대한 정보가 주어진 클래스에 대한 복잡도를 측량하기 위한 메트릭을 제안한다. 제안된 메트릭은 클래스들 사이의 상호작용으로 나타나는 협력의 복잡도와 인터페이스 복잡도를 포함하며, 개별 클래스의 복잡도를 사용하여 전체 시스템의 복잡도를 계산하도록 확장할 수 있다. 개발 생명주기의 가장 초기 단계인 분석단계에서 클래스를 어떻게 분할할 것인지에 대한 선택에 직면했을 때, 올바른 클래스 분할을 선택하도록 도와줌으로써, 나머지 개발 단계에서 개발에 요구되는 노력과 시간을 예측하고 관리하는데 사용될 수 있다.

1. 서론

객체지향 방법론에 대한 연구가 활발하게 진행되어 온 것과는 달리 객체지향 소프트웨어와 그 개발 절차의 품질을 향상시키기 위한 측량 방법은 널리 사용되지 못하였다. 그러나, 객체지향 프로그램의 효율성과 설계의 품질을 평가하기 위해서는 필수적으로 정량적 메커니즘을 사용해야 한다[1].

특히, 개발비용을 예측하기 위한 요구가 커질수록 소프트웨어 개발 생명주기의 초기에 측량해야 하는 필요성은 더욱 강해진다. 따라서, 분석단계 산출물에 대하여 이루어지는 측량은 나머지 개발 주기 단계에서 발견될 수 있는 결함에 대한 예측이나, 시스템의 개발에 소요될 노력과 비용을 예측하는데 사용될 수 있으므로 그 중요성이 더욱 강조되고 있다.

본 논문에서는 클래스 기반(class-driven)의 분석모델링 절차에 따라 작성되는 모델을 사용하여, 클래스의 책임과 연관 및 상속 관계 정보가 주어진 클래스에 대한 복잡도를 측량하기 위한 메트릭을 제안한다. 이 메트릭은 소프트웨어 시스템의 복잡도가 개발 시간에 직접적으로 영향을 준다는 사실을 기반으로 정의하였으며, 소프트웨어 개발 노력 및 클래스 분할에 대한 선택이 필요한 경우에 적용할 수 있다.

2. 관련연구

객체지향 개발 방법은 기존의 절차적(procedural) 소프트웨어 개발 방법과 구별되는 두드러진 특징을 가지고 있기 때문에, 객체지향 소프트웨어의 메트릭을 이런 특징에 따라 계층적으로 분류할 수 있다[2].

(1) 시스템에 대한 메트릭.

한 시스템은 여러 컴포넌트들로 나누어질 수 있기 때문에, 컴포넌트의 좋은 특성은 시스템의 좋은 특성이 될 수 있으며, 그 역도 성립하게 된다.

(2) 결합도에 대한 메트릭.

클래스들은 부 시스템을 형성하기 위하여 다른 클래스들과 상호작용을 한다. 이러한 상호작용의 특성 때문에 많은 결합도나 많은 객체들의 사용으로부터 복잡도를 표현하게 된다.

(3) 상속성에 대한 메트릭.

클래스 구조를 표현하는 모델에 나타나는 클래스들을 클래스 계층구조라고 부르며, 이 클래스와 부모 클래스 사이의 상속 관계는 개발 과정에서 일어날 수 있는 변화나 개선이 이루어지는 범위를 알려주게 된다.

(4) 클래스에 대한 메트릭.

한 클래스는 메소드를 포함하고 있다. 메소드를 가지는 클래스는 프로그래밍 작업을 복잡하게 하는 이질적인 데이터를 가질 수 있다.

(5) 메소드에 대한 메트릭.

속성과 데이터 구조가 분명하게 잘 이해되도록 정의되어

있는 반면, 메소드는 보통 구조적 프로그래밍에서의 프로시쥬어(procudures)와 매우 흡사하게 개발된다.

위와 같이 객체지향 메트릭을 분류할 수 있으며, 아래의 <표 1>은 지금까지 제안되어 온 객체지향 소프트웨어의 메트릭을 분류한 것이다. 이들 기존에 제안되었던 메트릭들의 문제점은 모두 설계에 대한 메트릭스를 제안하고 있으나, 구현단계 이후에 적용할 수 있는 메트릭스까지를 포함하고 있다는 것이다. 즉, 구현이 끝난 소스코드에 대한 측량이라고 볼 수 있다. 또한, 기본적인 개념이나 정의에 대한 설명만을 하고 있으며, 구체적으로 적용할 수 있는 방법을 설명하지 못하고 있다[3][4].

이에 본 논문에서는 분석단계에서 식별되는 클래스와 클래스의 책임 및 관계에 대한 정보만을 가지고 측량할 수 있는 메트릭을 정의하였다. 이 메트릭은 이해하기 쉬운 수학적 명세를 가지고 있으며, 메트릭의 정의에 대한 가설과 제한점을 포함하고 있다.

3. 복잡도 메트릭 CM(Comlexity Metric)

3.1 기본적인 정의

복잡도 메트릭 CM을 정의하기 위하여 먼저, 다음과 같은 항목을 정의하였다.

정의 1. 주어진 문제 영역(domain)은 식별가능하고 상호작용을 하는 물리적 또는 추상적 개념인 개체로 구성되어 있다. 개체들은 객체지향 분석 단계에서 클래스로 대응된다.

정의 2. 각각의 클래스는 자신이 제공하는 일련의 서비스들에 의하여 설명된다. 이러한 일련의 서비스를 책임(responsibilities)이라고 한다.

정의 3. 한 책임을 수행하기 위하여, 한 클래스가 다른 클래스와 연관 또는 구성(aggregation)관계에 있다면, 이 관계를 협력(collaborations)이라고 부른다.

	시스템	결합도	상속성	클래스	메소드
Abreu	SC1 SR1 SR2 SR3			CC2 CC3 CR1 CR2 CR3	
Chen		CCM OCM	CHM	OXM RM CM ACM	
Chidamber		CBO	DIT NOC	WMC RFC LOCM	
Lee	HC PC			CC	MC
Li		MPC		DAC NOM Size2	Size1
Moreau		GSDM	IL		SSM MCC
Sharble		NOT VOD		WAC	
Williams	CBC CSC	CCR COU			

<표 1> 객체지향 메트릭의 분류

정의 4. 한 클래스와 협력하는 클래스들을 협력자라고 부른다.

정의 5. 클래스는 부모 클래스로부터 책임을 상속 받을 수 있다. 상속 받은 책임은 자녀 클래스의 인터페이스의 일부분이 된다.

위의 정의와 함께, 다음은 클래스의 책임과 협력자의 수에 대한 가설이다.

가설 1. 상속 받지 않은 책임과 협력자의 수가 많은 클래스는 그 수가 적은 클래스보다 개발하기가 좀더 복잡하다.

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

가설 3. 개발 복잡도는 상속 받지 않은 책임의 수와 협력자의 수가 증가하는 것에 대하여 비선형적(non-linearly)으로 증가한다.

가설 4. 사용상의 복잡도는 책임의 전체 개수가 증가하는 것에 대하여 비선형적으로 증가한다.

가설 5. 한 클래스는 항상 자기자신과 협력한다.

3.2 복잡도 척도의 유도

본 논문에서 제안하고 있는 복잡도 척도 CM은 위의 가설들을 기반으로 정의되었고, 두개의 항을 가지고 있다. 첫번째 항은 협력의 복잡도를 나타내고, 두 번째 항은 인터페이스 복잡도를 나타낸다. 즉, CM의 공식은

CM = 협력의 복잡도 + 인터페이스 복잡도

가 된다.

협력의 복잡도는 가능한 협력의 최대수를 나타내며, 구현되어지는 각각의 책임에 대하여 협력자들이 사용된다고 가정한다. 그러므로, 협력의 복잡도를 C1이라고 한다면,

$$C1(E) = r(1+c) \quad \cdots\cdots\cdots\cdots(식\ 1)$$

이 된다. (식 1)에서 r은 상속 받지 않은 책임의 수이고, c는 협력자의 수이다.

두 번째 항은 인터페이스 복잡도이다. 이것은 책임의 전체 개수 R로서 계산된다. 그러나 이것은 복잡도와 R 사이에 비선형 관계를 가정한 가설4에 위배된다. 따라서, 인터페이스 복잡도를 상수지수 m에 의해 증가되는 것으로 정의하였다. 계산의 편의를 위하여 m=2로 한다면, 인터페이스 복잡도는 R^2으로 계산된다. 따라서, 한 클래스의 인터페이스 복잡도를 C2라고 한다면,

$$C2(E) = \frac{1}{(1+c)}\left\{R^2 + \sum_{i=1}^{c} C2(E_i)\right\} \quad \cdots\cdots\cdots\cdots(식\ 2)$$

가 된다. (식 2)는 전체 R개의 책임을 가지고 있고, c개의 다른 클래스들 $E_1, E_2, \cdots, E_c$ 와 협력하는 클래스 E에 대한 인터페이스 복잡도이다. 그러므로, CM의 완전한 공식은 다음과 같다.

$$CM(E) = r(1+c) + \frac{1}{(1+c)}\left\{R^2 + \sum_{i=1}^{c} C2(E_i)\right\} \quad \cdots\cdots\cdots\cdots(식\ 3)$$

또한, 복잡도 메트릭 CM은 한 시스템을 표현하는 클래스들의 집합에 대한 전체 복잡도를 계산하도록 확장할 수 있다. 그러므로, 만약 시스템 S가 클래스들 $E_1, E_2, \cdots, E_n$로 이루어져 있다면, S의 복잡도는 아래 (식 4)와 같이 정의된다.

$$CM(S) = \frac{1}{n}\sum_{i=1}^{n} CM(E_i) \quad \cdots\cdots\cdots\cdots(식\ 4)$$

3.3 예제

복잡도 척도를 적용하기 위한 예로서, 간단한 오디오 시스템의 분석 모델[6]에 나타난 두 클래스"Speaker"와 "AudioController"가 있다. 클래스"Speaker"의 복잡도는 7이고, 클래스 "AudioController"의 복잡도는 39로 계산된다[부록A 참조]. 이로써, 복잡도가 높은 클래스에 대한 개발 노력이 요구된다는 것을 알 수 있게 된다.

또한, 클래스 분할과 같은 선택에 직면했을 때, 복잡도 척도를 이용하여 어떤 선택을 해야 할지 결정할 수 있다. [부록 B]의 경우, 두 가지 설계 방안의 복잡도는 a가 21.5이고, b가 19.2로 계산된다.

비록 클래스의 수가 많아지는 것이 유지보수의 노력과 비용을 요구한다[7]고 해도, 상속을 통한 재사용이 이루어지는 경우의 복잡도가 더 낮으므로, 설계 b를 선택하는 것이 더 효율적이라고 말 할 수 있다.

복잡도는 클래스나 클래스로 구성된 시스템을 개발하는데 드는 시간으로 정의되기 때문에, 복잡도가 낮은 클래스를 선택함으로써 개발에 드는 노력과 시간을 줄 일수 있게 된다.

4. 결론 및 향후연구과제

기존의 객체지향 분석단계 산출물을 통한 클래스의 속성과 행위, 관계 정보에 의존한 메트릭은 분석 초기에 적용하기가 어려웠다[8]. 본 논문에서는 이러한 점을 개선하여 식별된 클래스의 책임을 기반으로 하는 새로운 메트릭을 제시하였다. 이 메트릭은 연관이나 구성을 협력 관계로 정의하여, 협력자와 관계를 가지고 있을 경우에 대한 복잡도를 측량하는 것이다. 또한, 클래스의 복잡도는 단순히 속성의 개수나 행위의 개수에 국한된 것이 아니라, 그 클래스가 상호작용을 하는 다른 클래스의 인터페이스 복잡도와 관계가 있다는 것을 포함하고 있다.

제안된 메트릭은 클래스의 복잡도를 평가하여 높은 복잡도를 가진 클래스들에 대한 개발 노력과 유지보수 비용을 예측 하는데 사용될 수 있다. 또한, 개발자들이 클래스에 대한 설계에서 어떤 선택에 직면했을 때 유용한 길잡이가 될 것이다.

현재 제안된 복잡도 메트릭이 실제 시스템 개발에서 효용성이 있는지를 확인하기 위하여 C++클래스에 대한 복잡도 측량 도구를 개발하고 있다.

앞으로 개발된 측량 도구를 이용한 충분한 사례연구를 통하여, 제안된 메트릭의 유효성 검증이 이루어져야 하며, 실제 객체지향 프로젝트에 적용한 자료를 통한 실증적인 분석이 필요하다.

5. 참고 문헌

[1]. R. S. Pressman, " Software Engineering - A Practitioner's Approach", 4th ed. McGraw-Hill Comp., 1998.,

[2] Clark Archer, Michael Stinson, "Object-Oriented Software Maseures", Technical Report of CMU/SEI, 95-TR-002, April 1995.

[3] B. Henderson-Sellers, Object-Oriented Metrics : Measures of complexity, Prentice-Hall, 1996.

[4] 고현희, "품질예측 및 향상을 위한 객체지향 설계 메트릭스 및 품질 평가에 관한 연구", 숙명여자대학교 전산학과 석사학위논문, 2000년 2월.

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

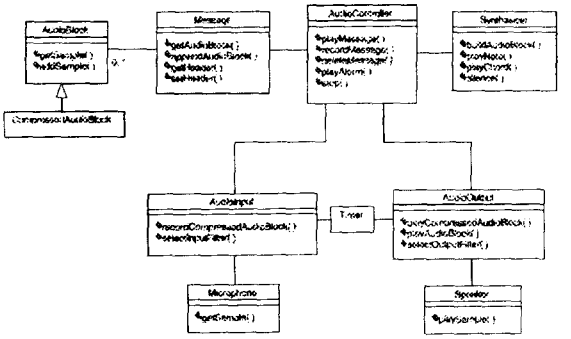
[6] Ivan P. P., Johan L., "Digital Sound Recorder : A case study on designing embedded systems using the UML", TUCS Technical Report No. 234, January 1999.

[7] S. R. Chidamber, C. F. Kemerer, "Towards a Metrics Suite for Object-Oriented Design", Proc. of 6th ACM conf. on Object-Oriented Systems, Languages and Applications, pp.197-211, 1991.

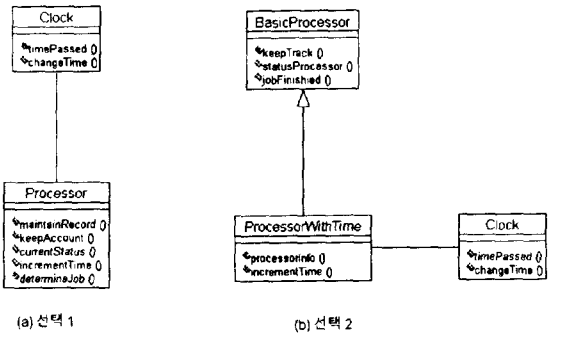
[8] 김유경, 고병선, 고현희, 박재년, "객체지향 소프트웨어의 품질 메트릭스에 관한 연구", 제 26회 한국정보과학회 춘계 학술발표 논문집, 제26권, 제2호, pp.629-631, 1999.

[부록]

A. 오디오 시스템의 클래스도



B. 클래스 분할에 대한 예제



(a) 선택 1 (b) 선택 2

(a) 선택 1에 대한 복잡도 계산

$CM(Clock) = 2(1+1) + 1/2(2^2 + 5^2) = 18.5$

$CM(Processor) = 5(1+1) + 1/2(2^2 + 5^2) = 24.5$

$CM(선택\ 1) = 1/2(18.5+24.5) = 21.5$

(b) 선택 2에 대한 복잡도 계산

$CM(Clock) = 2(1+1) + 1/2(2^2 + 5^2) = 18.5$

$CM(BasicProcessor) = 3(1+0) + 3^2 = 12$

여기에서, 클래스 "ProcessorWithTime"은 상속 받은 책임을 합한 전체 책임의 개수가 2+3으로 계산된다.

$CM(선택\ 2) = 1/2(18.5 + 12 + 27) = 19.2$