제18회 한국정보처리학회 추계학술발표대회 논문집 제9권 제2호 (2002.11)

분석단계에서 컴포넌트 인터페이스의 복잡성 측정

고병선° 박재년
컴퓨터과학과 숙명여자대학교
{kobs°, jnpark}@sookmyung.ac.kr

Measuring complexity of a component interface in the component analysis phase

Byung-Sun Ko° Jai-Nyun Park
Dept. of Computer Science, Sookmyung Women's University

요 약

독립적인 재사용 단위인 컴포넌트를 조립하여 시스템을 개발하는 컴포넌트 기반 시스템의 품질은 각 개별 컴포넌트의 품질에 영향을 받는다. 그러므로, 시스템의 품질 향상과 컴포넌트의 폭넓은 사용을 위해 개별 컴포넌트에 대한 품질 측정이 필요하다.

본 논문에서는 컴포넌트 분석단계의 정보를 사용하여 컴포넌트 인터페이스가 서비스를 제공하기 위해 얼마나 복잡한지를 측정한다. 이러한 독립적인 개별 컴포넌트의 인터페이스 복잡성 측정은 크기에 기반을 두며, 컴포넌트 기반 시스템 개발주기의 초기 단계인 컴포넌트 분석단계 산출물에 대해 이루어진다. 이러한 측정을 통해, 컴포넌트 개발에 대한 노력, 비용 및 시간, 결함 발생률, 사용 용이성 등을 예측할 수 있으며, 이로써 개발될 컴포넌트의 생산성 및 품질을 예측 및 관리하는데 사용할 수 있으므로, 그 중요성이 크다할 수 있다.

1. 서 론

컴퓨터 분야의 기술 발전은 규모가 크고 복잡하며, 요구사항의 변화에 유연하게 대응하는 소프트웨어의 개발을 요구하고 있다. 보다 빠른 시간안에 적은 비용으로 사용자의 변화하는 요구사항을 충족시키며 시스템을 개발하기 위해 소프트웨어 재사용 기술이 발전하게 되었다.

재사용 문제의 해결책으로 객체지향 개발(object-oriented development) 방법론이 사용되었으나, 클래스 내부의 데이터와 동작을 파악해야 하는 어려움으로 재사용의 한계가 나타나기 시작했다. 그후, 컴포넌트 기반 개발(component-based development) 방법론이 소프트웨어 개발에서의 독립성과 재사용 문제를 해결하기 위한 새로운 기술로 나타나기 시작했다. 기능의 조각으로써 보다 큰 재사용 단위인 컴포넌트를 조립함으로써 시스템을 개발하는 방법으로, 개발 시간과 비용을 줄이고 생산성을 향상시킬 수 있다.

독립적인 재사용 단위인 컴포넌트를 조립하여 시스템을 개발하는 컴포넌트 기반 시스템의 품질은 각 개별 컴포넌트의 품질에 영향을 받는다. 그러므로, 시스템의 품질 향상과 컴포넌트의 폭넓은 사용을 위해 개별 컴포넌트의 품질을 측정하는 것이 필요하다 [1, 2]. 측정은 개발할 시스템의 비용과 노력을 예측하도록 하여, 좀더 품질 좋은 시스템을 개발하도록 한다[3].

따라서, 본 논문에서는 컴포넌트 분석단계의 정보를 사용하여 컴포넌트 인터페이스가 서비스를 제공하기 위해 얼마나 복잡한지를 측정한다. 이러한 개별 컴포넌트의 복잡성 측정을 통해 개발 노력, 비용 및 시간, 결함 발생률, 테스팅 등을 예측 가능하여, 개발될 컴포넌트의 생산성 및 품질 예측 및 관리를 위해 사용할 수 있다.

2. 관련연구

2.1 컴포넌트 기반 개발 방법론

컴포넌트 기반 개발 방법론은 전통적인 개발 방법론의 연장선으로, 클래스 보다 큰 재사용 단위인 컴포넌트를 기능의 조각으로써 조립함으로써 시스템을 개발하는 방법이다. 소프트웨어도 하드웨어 부품처럼 조립이 가능해 개발자가 필요로 하는 컴포넌트를 구입해 설치만 하면 동작이 가능한 시스템을 구성할 수 있다는 것으로, 개발 시간과 비용을 줄이고 생산성을 향상시킬 수 있다. 또 다른 효과로 독립적인

컴포넌트의 조립으로 유지보수가 용이해짐에 따라, 분석 및 설계 단계에서 더 많은 시간과 노력을 들일 수 있기 때문에, 고품질의 소프트웨어를 기대할 수 있다[1, 2, 4, 5, 6].

컴포넌트 기반 개발 절차는 객체지향 개발 절차와 유사한 점이 많다. 반복적이며 점진적인 특성을 따르는 거시적 개발 절차와 요구사항 분석, 분석, 설계, 구현, 시험의 단계로 구성되는 미시적 개발 절차로 구성된다. 미시적 개발 절차 중 분석 단계에서는 컴포넌트와 인터페이스가 도출된다[2, 7]. 다시 말하면, 분석 단계에서는 컴포넌트의 구조 설계(component architecture design)와 인터페이스 정의(interface definition)가 이루어진다. 분석된 클래스들 중에서 가장 중요한 클래스를 중심으로 높은 응집력과 낮은 결합력을 갖도록 개별 컴포넌트로 분할을 한다. 그리고, 각 컴포넌트가 제공할 기능을 정의하는 인터페이스 정의 작업을 수행하는데, 이는 기능 수행을 위한 클래스들과의 협력 관계를 행위의 매개변수를 통해 표현한다. 이로써 컴포넌트의 구조와 행위가 파악된다.

2.2 컴포넌트의 특징

객체지향 기술은 실세계의 데이터와 이에 대한 처리를 캡슐화하여 클래스로 모델링하고, 이들간의 상호 관계를 최소로 줄여 재사용을 하고자 했다. 그러나, 재사용을 위해 클래스의 내부를 파악해야 하는 어려움으로, 객체지향 시스템의 클래스들을 그들의 상호 작용을 기반으로 하나의 기능을 제공하기 위한 클래스들의 모임인 컴포넌트로 묶어, 재사용하게 되었다.

특정 기능을 수행하는 컴포넌트는 기능 수행에 필요한 요소인 데이터와 동작으로써, 내부에 클래스들간의 상호 관계와 외부에 인터페이스(interface)로 구성되는 기능 독립성을 갖는다. 인터페이스는 컴포넌트의 기능을 추상화시키며, 소프트웨어 단위인 컴포넌트를 외부로 보이게 하는 역할을 하며, 내부의 클래스들에 의해 수행되는 서비스에 접근하기 위한 행위들(operation)의 집합이다[7]. 상호 의존성 있는 업무 기능과 데이터의 통합이라는 컴포넌트의 특성은 소프트웨어의 응집력을 높인다[2, 5, 8].

2.3 컴포넌트 메트릭스의 필요성

객체지향 시스템은 속성과 메소드가 캡슐화된 클래스를 기반으로 개발된다. 그러므로, 대부분의 객체지향 메트릭스는 객체지향 시스템의 기본 단위인 클래스에 기반을 두고, 클래스, 메소드, 상속성, 캡슐화 등을 고려하여 품질을 측정하였다[3].

그러나, 컴포넌트 기반 시스템은 하나 이상의 컴포넌트로 이루어지며, 또한 컴포넌트는 하나 이상의 클래스와 인터페이스로 구성되므로, 객체지향 메트릭스를 그대로 컴포넌트에 적용하는 것은 부적당하다. 그러므로, 상호작용이 이루어지는 컴포넌트 내부의 클래스들과 컴포넌트 인터페이스와 같은 추가적인 정보를 고려한 새로운 컴포넌트 메트릭스가 필요하다.

3. 컴포넌트 인터페이스 복잡성

3.1 컴포넌트에 대한 기본적 정의

컴포넌트에 대한 정의는 여러가지가 있으나, 분석 단계의 컴포넌트에 대해 인터페이스의 복잡성을 측정하기 위해, 컴포넌트의 특성에 대해 아래와 같이 정의한다.

정의 1. 컴포넌트는 자신만의 특정 서비스를 제공하는 기능 단위이다.

정의 2. 컴포넌트는 하나 이상의 클래스들로 구성된다.

정의 3. 컴포넌트가 제공하는 서비스는 인터페이스에 의해 표현되며, 또한 컴포넌트의 서비스 사용은 인터페이스를 통해 접근된다.

정의 4. 컴포넌트는 자신의 특정 서비스를 제공하기 위해 클래스와 인터페이스 사이에 상호작용을 하게 된다.

다음은 컴포넌트의 구성요소인 클래스와 인터페이스에 대한 가설로 아래와 같다.

가설 1. 많은 수의 클래스로 구성된 컴포넌트는 적은 수의 클래스로 구성된 컴포넌트에 비해 개발하기가 복잡하다.

가설 2. 인터페이스가 많은 컴포넌트는 인터페이스가 적은 컴포넌트에 비해 사용하기가 복잡하다.

가설 3. 많은 수의 행위(operation)로 구성된 인터페이스는 적은 수의 행위로 구성된 인터페이스에 비해 개발하기가 복잡하다.

가설 4. 매개변수가 많은 행위는 매개변수가 적은 행위에 비해 사용하기가 복잡하다.

3.2 컴포넌트 인터페이스의 복잡성 측정

컴포넌트는 하나의 서비스를 제공하기 위한 클래스들의 묶음으로, 그 서비스는 인터페이스로 표현된다. 인터페이스는 내부의 클래스들에 의해 수행되는 서비스에 대한 축약 표현이 되며, 또한 컴포넌트 내부의 여러 클래스에 공통적으로 접근하기 위한 통로 역할을 하게 된다[1, 7].

특정 서비스를 제공하기 위해 컴포넌트는 인터페이스를 통해 기능적으로 컴포넌트 내부의 클래스들과 협력 관계를 갖게 된다. 구체적으로 인터페이스는 일종의 메소드인 행위(operation)들을 통해 컴포넌트 내부의 클래스나 다른 컴포넌트와 협력한다.

다시 말하면, 컴포넌트의 서비스 제공은 행위가 매개변수를 통해 컴포넌트 내부의 클래스와 협력을 함으로써 제공된다.

한 컴포넌트가 서비스를 제공하기 위해 기능적으로 어떤 협력 관계를 얼마나 많이 가지는지를 정확하게 알 수는 없지만, 제공되는 인터페이스를 통해 컴포넌트의 사용이 얼마나 복잡할지를 예측할 수는 있다. 인터페이스는 컴포넌트 서비스의 사용 및 접근에 대한 통로 역할을 하므로, 너무 복잡하지 않아야 사용하기가 좋을 것이다. 컴포넌트 인터페이스가 너무 복잡하다면, 컴포넌트의 사용 및 개발에도 많은 노력과 어려움이 필요할 것이다.

컴포넌트 인터페이스 복잡성은 두가지 측면에서 측정될 수 있다.

첫째는, 행위의 수로 계산 가능하다. 행위는 일종의 메소드로 컴포넌트 내부의 클래스나 다른 컴포넌트와 협력하기 위한 통로 역할을 하는 인터페이스의 구성요소이다..

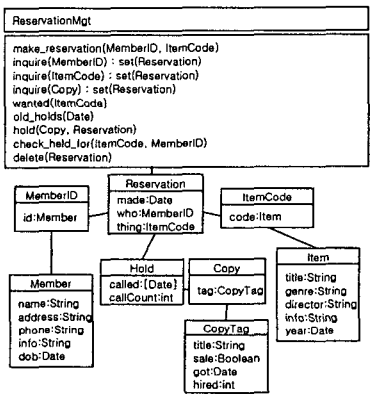
행위에 의한 컴포넌트 인터페이스 복잡도(Component Interface Complexity by Operations)를 *CICO*라 표현한다면, $CICO(Co) = o$ 이다. 여기서, o는 인터페이스의 전체 행위의 수이다. 이는 위의 가설 3을 기반으로 정의되었으며, 행위의 수가 많은 인터페이스는 기능 수행 과정이 복잡하며, 컴포넌트의 사용 및 개발이 어려움을 의미한다.

둘째는, 행위의 매개변수의 수로 계산 가능하다. 인터페이스의 행위는 매개변수를 통해 컴포넌트 내부의 클래스들과 협력 관계를 갖게 된다.

매개변수에 의한 컴포넌트 인터페이스 복잡도(Component Interface Complexity by Parameters of operations)를 *CICP*라 표현한다면, $CICP(Co) = p$ 이다. 여기서, p는 행위의 전체 매개변수의 수이다. 이는 위의 가설 4를 기반으로 정의되었으며, 많은 매개변수로 구성된 행위들을 갖는 인터페이스는 기능 수행 과정이 복잡하며, 컴포넌트의 사용 및 개발이 어려움을 의미한다.

컴포넌트의 서비스는 인터페이스가 메소드 형태인 행위를 통해 컴포넌트 내부의 클래스들과의 협력을 통해 제공된다. 그러므로, 특정 기능 수행을 위한 컴포넌트 인터페이스의 복잡성 측정의 인자로 인터페이스의 행위를 택했다. 또한, 행위의 협력관계는 구체적으로 매개변수를 통해 이루어지므로, 매개변수 또한 복잡성 측정의 인자로 택했다. 따라서, 제안한 메트릭 *CICO*와 *CICP*는 행위가 많을수록, 많은 매개변수로 구성된 행위일수록 컴포넌트 인터페이스의 사용 및 개발이 복잡해진다는 것을 나타낸다.

제안된 메트릭의 측정값 계산을 위한 사례연구로 택한 비디오 대여 시스템[9]은 회원관리, 대여, 예약의 세가지 컴포넌트로 구성된다. 컴포넌트의 분석 단계에서는 클래스와 인터페이스가 도출되어 컴포넌트의 구조와 행위가 파악되는데, 그 중 Reservation Manager 컴포넌트의 분석단계 산출물은 [그림 1]과 같으며, ReservationMgt 인터페이스의 행위들과 관련된 여러 클래스들로 구성된다. 이때, 행위에 의한 컴포넌트 인터페이스 복잡도(*CICO*)는 9이고, 매개변수에 의한 컴포넌트 인터페이스 복잡도(*CICP*)는 12이다.



[그림 1] Reservation Manager 컴포넌트의 구성 (클래스와 인터페이스)

4. 평가 및 증명

제안된 메트릭스의 적합성을 이론적 평가와 컴포넌트의 크기에 대한 수학적 평가로 증명한다.

4.1 크기 메트릭 속성에 대한 평가

제안된 메트릭스는 인터페이스의 크기에 대한 메트릭스로 이론적 평가는 Briand[10]가 제안한 크기(size) 메트릭이 만족해야만 하는 필요조건적 속성에 적용하여 평가해 본다.

성질 1. Non-negativity

인터페이스는 기능 수행을 위해 행위로 구성되고, 또한 행위는 내부의 클래스를 매개변수로 사용하므로, 측정값은 최악의 경우 0일 수는 있으나, 음수는 아니므로, 성질 1은 만족된다.

성질 2. Null Value

만일, 인터페이스를 구성하는 행위가 없다면, 매개변수도 없으므로, 측정값은 0이다. 그러므로, 성질 2도 만족된다.

성질 3. Module Additivity

인터페이스는 기능 수행을 위한 여러 행위로 구성된다. 만일, 행위가 서로 공통 부분이 없는 두 요소

로 구분된다면, 인터페이스의 전체 크기는 행위들의 두 요소의 합과 동일하므로, 성질 3은 만족된다.

4.2 컴포넌트 크기에 대한 증명

컴포넌트는 특정 서비스를 수행하는 기능의 단위인데, 설계자들은 너무 커서 다루기 어려운 컴포넌트나 재사용을 하기에 너무 작은 컴포넌트의 설계는 좋지 않다고 생각한다. 즉, 컴포넌트는 기능의 단위로 너무 크지도 또한 너무 작지도 않아야 한다.

따라서, 제안된 컴포넌트 인터페이스 복잡성 측정이 컴포넌트의 분할과 결합에 대해 어떻게 반응하는가에 대해 증명한다. 제안된 메트릭스는 둘 다 크기에 관한 메트릭스이므로, 그 중 행위에 의한 컴포넌트 인터페이스 복잡도 메트릭스에 대해서만 살펴보겠다.

먼저, 컴포넌트의 분할에 대해 살펴본다. c개의 클래스로 구성되고, 인터페이스의 전체 행위의 수는 o개인 컴포넌트 Co가 c_1과 c_2개의 클래스와 o_1과 o_2개의 행위로 구성되는 인터페이스를 갖는 두 컴포넌트 Co_1과 Co_2로 분할되었다고 하자. 컴포넌트의 분할은 보다 작은 기능 단위로의 분할을 의미하므로, 기능의 축소로 인해 기본 행위의 수보다 감소할 수 있으므로, $o \geq o_1 + o_2$ 이다. 이때, 컴포넌트 인터페이스 복잡도 계산은 아래와 같으며, 이는 규모가 큰 하나의 컴포넌트가 적당한 크기의 두 컴포넌트로 분할될 경우 좀더 낮은 복잡도를 갖게 된다는 것을 나타낸다.

$$\begin{aligned} CICO(Co) &= o \\ &\geq o_1 + o_2 \\ &= CICO(Co_1) + CICO(Co_2) \end{aligned}$$

다음은 컴포넌트의 결합에 대해 생각해 본다. o_1과 o_2개의 행위로 구성되는 인터페이스와 c_1과 c_2개 클래스로 구성되는 두 컴포넌트 Co_1과 Co_2가 새로운 컴포넌트 Co로 결합되었다고 하자. 새로운 인터페이스는 o_1과 o_2개의 행위를 결합한 것인데, 이때 행위들 사이에 중복이 발생할 수 있으므로, $o_1 + o_2 \geq o$이 된다. 또한, 결합되는 클래스들 사이에도 중복이 발생할 수 있으므로, $c_1 + c_2 \geq c$이 된다. 이때, 컴포넌트 인터페이스 복잡도 계산은 아래와 같으며, 서로 비슷한 기능을 하는 두 컴포넌트를 하나의 컴포넌트로 결합할 경우 좀더 낮은 복잡도를 갖게 된다는 것을 나타낸다.

$$\begin{aligned} CICO(Co_1) + CICO(Co_2) &= o_1 + o_2 \\ &\geq o \\ &= CICO(Co) \end{aligned}$$

5. 결론 및 향후 연구과제

컴포넌트 분석단계의 정보를 사용하여 컴포넌트 인터페이스의 복잡성을 행위 측면과 매개변수 측면에서 측정하였다. 이러한 개별 컴포넌트 인터페이스의 복잡성 측정은 개발을 위한 비용, 노력, 시간, 개발시 결함 발생률, 사용을 위한 노력과 어려움 등을 예측하는데 사용 가능하다. 또한, 이는 시스템 개발 주기의 초기에 이루어지므로, 앞으로 진행될 시스템 개발 시간과 비용을 줄이고 생산성을 향상시킬 수 있다.

향후 연구과제는 제안된 측정이 많은 사례연구를 통해 실험적 유효성 검증이 이루어져야 하며, 평가를 위한 기준값 도출이 필요하다.

참고문헌

[1] Colin Atkinson, Joachim Bayer, Christian Bunse, Erik Kamsties, Oliver Laitenberger, Roland Laqua, Dirk Muthig, Barbara Peach, Jurgen Wust, Jorg Zettel, Component-Based Product Line Engineering with UML, Addison Wesley, 2002.

[2] George T. Heineman, William T. Councill, Component-Based Software Engineering : Putting the Pieces Together, Addison-Wesley, 2001.

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

[4] Clemens Szyperski, Component Software: Beyond Object-Oriented Programming, ACM Press and Addison-Wesley, 1998.

[5] Roger S. Pressman, Software Engineering : A Practitioner' s Approach, Fifth Edition, McGraw-Hill, June 2000.

[6] 한국소프트웨어 컴포넌트 컨소시엄, 컴포넌트란 무엇인가? 알기쉬운 소프트웨어 컴포넌트, December 2001.

[7] John Cheesman, John Daniels, UML Components: A Simple Process for Specifying Component-Based Software, Addison-Wesley, 2001.

[8] 고병선, 박재년, "컴포넌트의 응집성 측정", 한국정보처리학회 논문지 D, Vol.9-D, No.4, pp. 613-618, August 2002.

[9] Desmond F. D'Souza, Alan C. Wills, Object, Component and Framework with UML : The Catalysis Approach, Addison-Wesley, 1999.

[10] Lionel Briand, Sandro Morasca, Victor Basili, "Property-based Software Engineering Measurement", IEEE Transactions on Software Engineering, Vol.22, No.1, pp.68-86, January 1996.