소프트웨어 모듈성을 정량적으로 측정하는 방법

(A method for quantitative measurement of software moduleness)

정필수[+] (Pilsu Jung) 안종선[++] (Jongsun Ahn) 박태현[+++] (Teahyun Park) 강성원[++++] (Sungwon Kang)
은나래[§] (Narae Eun) 고상원[¶] (Sangwon Ko)

요 약 소프트웨어의 품질을 평가하기 위해, 구성 모듈의 품질을 측정하는 것이 중요하다. 특히, 현업에서는 품질이 높은 모듈을 재사용하고 품질이 낮은 모듈을 개선하는 활동이 중요하기 때문에 모듈의 품질을 정량적으로 측정할 필요가 있다. 본 논문에서는 모듈이 지녀야 할 속성으로서 모듈성(moduleness)을 정의하고 모듈성을 구성하는 여러 가지 품질 속성과 품질 측면 및 측정 지표를 정의한다. 그리고 이들을 활용하여 모듈성을 정량적으로 측정하는 방법을 제안한다. 사례 연구에서는 C, C++, Java기반의 오픈소스 모듈들을 대상으로 모듈성을 정량적으로 측정하고 그 측정 결과를 검증함으로써 제안 방법의 효용성을 보인다.

키워드 소프트웨어 모듈성, 소프트웨어 품질 측정, 가중치 결정 방법, AHP

Abstract We need to measure the quality of each module to assess software quality. Especially, in industry, it is important to measure quality of modules quantitatively in order to improve bad modules and reuse good modules. In this paper, we define moduleness as characteristic that modules should be possessed and propose a method to measure software moduleness quantitatively. A case study conducted in this paper shows the usefulness of the method by measuring and evaluating moduleness of modules based on open source software developed by C, C++ and Java languages.

Key words Software moduleness, Software quality measurement, weighting method, Analytic hierarchy process

1. 서 론

소프트웨어 활용분야가 점점 넓어지면서 소프트웨어 품질은 중요한 이슈가 되었다[1]. 그 이유는 소프트웨어의 규모와 복잡도는 점점 증가하고 소프트웨어가 적용되는 도메인은 시간이 지남에 따라 다양해고 있기 때문이다. 현업에서는 레거시 소프트웨어에서 품질이 좋은 모듈을 재사용하여, 적은 비용으로 품질이 좋은 소프트웨어를 생산하기를 원한다. 또한, 품질이 나쁜 모듈들을 개선하여 해당 모듈이 사용된 소프트웨어의 전체 품질을 향상시키기를 원한다. 이를 위해, 모듈의 품질을 측정하여 평가함으로써 모듈의 현 수준을 평가할 수 있는 연구가 필요하다.

[+] 학생회원 : 한국과학기술원 전산학과 psjung@kaist.ac.kr
[++] 학생회원 : 한국과학기술원 전산학과 jsahn@kaist.ac.kr
[+++] 학생회원 : 한국과학기술원 전산학과 danapark@kaist.ac.kr
[++++] 종신회원 : 한국과학기술원 전산학과 교수 sungwon.kang@kaist.ac.kr
[§] 비회원 : LG전자 소프트웨어 아키텍처팀 narae.eun@lge.com
[¶] 비회원 : LG전자 소프트웨어 아키텍처팀 sangwon.ko@lge.com

소프트웨어 품질(software quality)이란 소프트웨어에 요구되는 품질 속성(quality attribute)들을 잘 반영하는 정도를 의미한다[1]. 품질 속성은 소프트웨어가 이해 관계자들의 요구를 얼마나 잘 반영하는지 나타내기 위해 사용되는 측정가능한 소프트웨어 특성이며[2], 이는 여러 가지 관련 품질 측면(quality concern)으로 구성된다. 품질 측면은 소프트웨어 품질 속성을 명세, 측정, 평가하기 위한 수단이다[3].

과거에 소프트웨어 품질을 정량적으로 측정하는 연구들이 많이 수행되었다[4,5,6]. 이들은 소프트웨어 품질을 여러 가지 품질 측면들로 나누고 각 품질 측면들을 측정하는 방법을 제시한다. 그러나 품질 측면들을 종합하여 소프트웨어의 수준을 평가하는 방법을 제시하지 않았기 때문에 현업에서 이를 활용하기 어렵다는 한계가 있다. 모듈로 구성된 소프트웨어의 모듈화 수준을 정량적으로 측정하기 위한 연구[7,8,9]도 수행되었다. 이들은 모듈화 수준을 평가하기 위한 품질 측면을 정의하고 그 측정 방법을 제안한다. 그러나 이들도 품질 측면들을 종합하여 평가하는 방법은 제시하지 않거나[8,9], 각 품질 측면들에 대한 가중치를 고려하지 않고 명확한 근거 없이 동일한 비중으로 더하여 계산하는 데 한계점을 지닌다[7]. 따라서 기존 연구들의 제안 방법을 활용하여 소프트웨어 모듈의 품질을 정량적으로 측정하기 어렵다.

본 논문은 소프트웨어 모듈성(moduleness)을 구성하는 여러 가지 품질 속성들과 품질 측면들을 정의하고 이들을 정량적으로 측정하는 방법을 제안한다. 제안 방법을 통해, 모듈의 현 수준을 객관적으로 파악할 수 있고, 객관적 수치를 활용하여 해당 모듈을 새로운 소프트웨어 개발에 재사용할지 또는 새롭게 개발할지를 결정하는 기준을 세울 수 있다. 또한, 재사용 가능하다면 추가적인 개선이 필요한지의 여부를 정량적 수치를 활용하여 결정할 수 있다.

본 논문의 구성은 다음과 같다. 제2절에서는 소프트웨어 품질을 측정하는 기존 연구들을 소개한다. 제3절에서는 모듈성을 정량적으로 측정하는 방법을 제시하고 제4절에서는 오픈소스를 활용한 사례 연구를 통하여 제안 방법의 효용성을 보인다. 끝으로, 제5절에서는 제안 방법의 기여와 향후 연구 방향에 대해 논의한다.

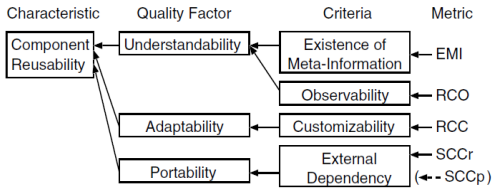
2. 소프트웨어 품질 평가에 관한 기존 연구

과거에 소프트웨어의 품질을 평가하기 위해 품질 속성을 구성하는 여러 가지 품질 측면들을 정의하고 이들을 정량적으로 측정하는 연구들이 수행되어 왔다. 과거 연구들은 객체지향 설계 품질을 측정하기 위한 연구, 소프트웨어 모듈화 수준을 측정하는 연구, ISO/IEC 9126 표준 또는 ISO/IEC 25010 표준에서 정의한 품질 속성을 측정하는 연구로 분류할 수 있다.

객체지향 설계 품질을 측정하는 연구는 Li et al.[10]의 연구와 Bansiya et al.[11] 연구가 있다. Li et al.[10]은 객체지향 소프트웨어 설계품질을 측정하기 위해 기존에 연구된 여러 메트릭을 분석하고 새로운 메트릭들을 추가하여 두 개의 상용 프로그램으로부터 효용성을 검증한다. Bansiya et al.[11]은 객체지향 설계에서 여러 가지 품질 속성을 측정하기 위한 계층 모델을 제안한다. 계층 모델은 품질 속성(quality attribute) 계층, 객체지향 설계 특성(design property) 계층, 객체지향 메트릭 계층, 객체지향 컴포넌트 계층으로 구성된다. 그리고 품질 속성과 설계 특성, 메트릭들을 정의하여 여러 가지 메트릭들의 측정값을 합산함으로써 품질 속성을 측정하는 방법을 제시한다. 그러나 측정값을 종합하여 평가하는 방법을 제시하고 있지 않기 때문에 소프트웨어의 객체지향 설계 품질이 어느 수준을 지니고 있는지 평가하기 어렵다.

소프트웨어 모듈화 수준을 측정하는 연구는 Sarkar et al.[8,9]의 연구와 Bangare et al.[4]의 연구 등이 있다. Sarkar et al.[8,9]은 모듈화 수준을 높이기 위한 여덟 가지 모듈 설계 원칙을 제안하고 각 설계 원칙을 정량적으로 측정할 수 있는 메트릭을 정의한다. 그리고 각 메트릭들을 실험을 통해 효용성을 검증한다. Bangare et al.[4]은 소프트웨어 유지보수성을 높이고 개발자들의 부담을 줄이기 위한 목적으로, 객체지향 모듈화 수준 측정 방법을 세 단계로 제안한다. 이를 수행하기 위해 네 가지 메트릭을 정의하고 각 메트릭을 측정하기 위한 알고리즘을 제안한다. 그리고 각 메트릭들을 실험을 통해 효용성을 검증한다. 두 연구는 제안한 메트릭을 종합하여 모듈화 수준을 평가하는 방법을 제시하고 있지 않기 때문에 제안한 메트릭 측정값을 통해 모듈화 수준을 주관적으로 판단해야 된다는 단점을 지닌다.

ISO/IEC 9126, 25010 표준에서 정의한 소프트웨어 품질 속성을 측정하는 연구는 상당히 많이 수행되었다. Washizaki et al.[6]은 컴포넌트 기반 개발(Component-based Development)에서 소프트웨어 컴포넌트의 재사용성(reusability)을 측정하기 위한 메트릭들을 제안한다. 컴포넌트는 경우에 따라 목적 코드(object code)만 사용자에게 공개되고 소스코드는 숨겨지는 경우가 많기 때문에 그들은 소스코드를 분석하지 않고 컴포넌트의 재사용성을 측정하기 위한 모델을 [그림 1]과 같이 제안한다. 이들은 재사용성을 세 가지 품질 요소(quality factor)와 네 가지 기준으로 나누어 각각에 대한 메트릭을 정의한다. Subramanian et al.[12]은 ISO/IEC 25010 표준에서 정의한 소프트웨어 적응성(adaptability)을 측정하기 위한 메트릭을 제안한다. 적응성을 측정하기 위해 컴포넌트와 커넥터가 각각 다른 컴포넌트와 커넥터에 적응 가능한 상태인지 판별하기 위한 아키텍처 수준의 적응성 메트릭을 정의하고 이를 이용하여 소프트웨어 수준의 적응성 메트릭을 정의한다. 이와 달리, Liu et al.[13]은 이해관계자들에 의해 발생되는 변경 요구사항에 대해 얼마나 영향을 많이 받는지 정량적으로 측정하여 소프트웨어 아키텍처의 적응성을 측정한다. 변경 요구사항에 대한 영향을 측정하기 위해서 발생 가능한 여러 가지 시나리오를 정의하고 각 시나리오에 대해 컴포넌트와 커넥터의 변경 정도를 측정한다. 세 연구에서 정의한 각 메트릭은 소프트웨어 품질에 영향을 주는 정도가 다를 수 있기 때문에 각 메트릭의 가중치를 결정할 필요가 있다. 그러나 이를 고려하지 않고 있다는 점에서 한계를 지닌다.



[그림 1] 컴포넌트 재사용성의 품질 속성 모델[6]

3. 모듈성을 정량적으로 측정하는 방법

본 절에서는 본 연구에서 다루는 모듈과 모듈성을 3.1절에서 정의하고 모듈성 품질 모델을 3.2절에서 제시한다. 이를 활용하여 3.3절에서는 모듈성을 정량적으로 측정하는 방법을 소개한다. 그리고 모듈성이 올바로 측정되었는지 검증하는 방법을 3.4절에서 소개한다.

3.1 모듈과 모듈성의 정의

기존의 많은 연구에서 모듈의 특성에 대해 언급하고 있다. Clements 등은 모듈을 관련된 행위, 지식, 의사결정 등을 제공하는 소프트웨어 구현 단위[14]라고 표현한다. Briand 등과 Sarkar등은 모듈은 시스템 내의 일부 코드 조각, 자료구조, 또는 부분 시스템 등으로 정의될 수 있다고 말하며[8,15], Rozanski등은 시스템을 개발하기 위해 고려

될 수 있는 기본 단위라고 말한다[16]. Hofmeister 등은 모듈의 내부 데이터와 함수들은 캡슐화 되어야 하며, 외부에 서비스를 제공하기 위해서 인터페이스를 정의해야 한다고 말한다[17]. 이들의 주장을 바탕으로 모듈을 다음과 같이 정의할 수 있다.

모듈의 정의: 기능을 제공하기 위한 관련된 함수와 클래스들의 집합으로 캡슐화된 소프트웨어 구현 단위

모듈의 정의에 따라, 모듈은 함수와 클래스들이 모여 이뤄진 파일 또는 디렉토리 단위로도 구성될 수 있다. 모듈이 가지는 관련된 여러 기능들은 함수와 클래스의 집합으로 구현될 수 있는데, 이들은 캡슐화되어 인터페이스를 통해서만 외부로부터 접근 가능해야 한다.

새로운 소프트웨어 개발할 때, 기존 모듈들을 활용하기 위해서 기존 모듈들이 모듈로서 자격을 갖추고 있는지 평가할 필요가 있다. 그 이유는 좋은 모듈을 재사용하면 그만큼 좋은 소프트웨어를 산출할 수 있기 때문이다. 즉, 잘 구성된 모듈을 재사용하면 소프트웨어 개발 비용을 줄일 수 있고 그 소프트웨어는 유지보수가 용이하며 결함 발생률이 낮다. 본 논문에서는 모듈이 얼마나 잘 구성되었는지 품질을 나타내는 척도로서 모듈성을 다음과 같이 정의한다.

모듈성(moduleness)의 정의: 모듈외부로부터 정보를 숨기고 다른 모듈과 독립적으로 기능을 수행함으로써 모듈의 기능을 여러 시스템에서 재사용하기 용이한 정도

3.2 모듈성의 품질 모델

소프트웨어 모듈성을 정량적으로 측정하기 위해, 먼저 모듈성을 구성하는 품질 속성과 품질 측면을 정의할 필요가 있다. 많은 연구들에서 모듈과 관련하여 여러 가지 품질 속성들을 제시하고 있지만, 이들을 모두 고려하는 것은 많은 노력이 필요하고 복잡하기 때문에 사용하기 어렵다. 또한, 모듈성과 상대적으로 관련이 적은 속성들까지 모두 고려하기 보다 관련이 깊은 속성들을 신중히 고려하는 것이 효과적이다. 본 논문에서는 소프트웨어 모듈에 관해 기술하는 서적[14,16,17,18,19]과 논문[20,21]들을 조사하였다. 그 결과, 여러 연구에서 여러 번 언급되는 모듈의 특성들을 다음 다섯 가지로 정리할 수 있다.

특성 1. 모듈은 관련된 기능들을 제공한다[14,16,17].
특성 2. 모듈은 외부로부터 내부 정보를 노출시키지 않는다[17,18].
특성 3. 모듈은 재사용이 용이해야 한다[16,20].
특성 4. 모듈은 이해하기 쉬워야 한다[19,21].
특성 5. 모듈이 지닌 기능을 시스템에 맞게 잘 적응할 수 있는 능력을 가져야 한다[20].

위 다섯 가지 특성을 바탕으로 MOOD (Model for Object-Orient Design)와 ISO/ IEC9126, ISO/ IEC25010 표준에서 정의한 품질 속성들을 참고하여 모듈이 갖춰야 할 품질 속성들로서 캡슐화(encapsulation)(특성 1, 2에 의해), 맞춤성(customizability)(특성 3, 5에 의해), 이해용이성(understandability)(특성 3, 4에 의해), 그리고 코드의 품질 수준을 평가하기 위해 코드 품질(code quality)을 정의한다. 각 품질 속성의 정의는 <표 1>과 같다.

<표 1>에서 정의한 네 개의 품질 속성들을 측정하기 위해 코드 수준에서 측정할 수 있는 품질 측면들을 정의할 필요가 있다. 본 논문에서는 CK(Chidamber & Kemerer) metrics[22]와 QMOOD [11]에서 제시하는 품질 측면들을 참고하여 모듈성의 네 품질 속성들과의 연관성을 분석한다.

모듈의 캡슐화는 정의와 같이 내부 정보를 숨기는 정도를 나타내는 정보 은닉(information hiding)이 갖춰져야 한다.

모듈의 맞춤성을 높여 다른 시스템에 쉽게 적용하기 위해 다른 모듈과의 의존성을 제거해야 한다.

그 이유는 새로운 시스템에 적용하기 위한 해당 모듈의 수정이 다른 모듈의 수정을 요구하여 추가적인 노력이 들기 때문이다. 또한, 모듈의 맞춤성을 높이기 위해 다른 시스템에 재사용이 용이해야 한다. 객체지향의 주요 특성인 상속과 다형성은 시스템의 재사용성을 높여 준다[23]. 따라서 모듈에 상속과 다형성의 원리를 적용하여 새로운 시스템에 적용하기 유용하게 만들 수 있다. 결론적으로, 맞춤성을 결정하는 품질 측면으로 결합도, 상속성, 다형성을 선정한다.

<표 1> 모듈성을 구성하는 품질 속성

품질 속성	정의
캡슐화	공개가 불필요한 모듈의 데이터와 함수를 숨겨 외부에서 보이지 않게 하는 정도
맞춤성	모듈이 새로운 시스템에 적용되기 위해 수정하기 용이한 정도
이해 용이성	사용자가 모듈 또는 소스코드의 구성과 의미를 파악하기 용이한 정도
코드 품질	코드가 잠재적 결함 발생률을 낮추고 유지보수를 용이하게 해줄 수 있는 정도

모듈의 기능을 쉽게 이해되도록 구성하기 위해서는 모듈을 구성하는 함수의 크기가 너무 크거나 내부 구현이 복잡하지 않아야 한다. 또한, 한 모듈의 기능과 관련된 데이터와 그 구현들끼리 잘 집적되어야 해당 모듈의 기능을 빠르게 이해하는데 도움이 된다. 그리고 모듈의 상속 깊이(inheritance depth)가 커서 추상화 수준이 높으면 해당 모듈을 이해하기 위해 여러 부모 클래스를 모두 분석해야 하므로 이해하기 어려워진다. 따라서 이해 용이성을 결정하는 품질 측면으로 평균 함수(메소드) 라인 수, 복잡도, 응집도, 추상성을 선정한다.

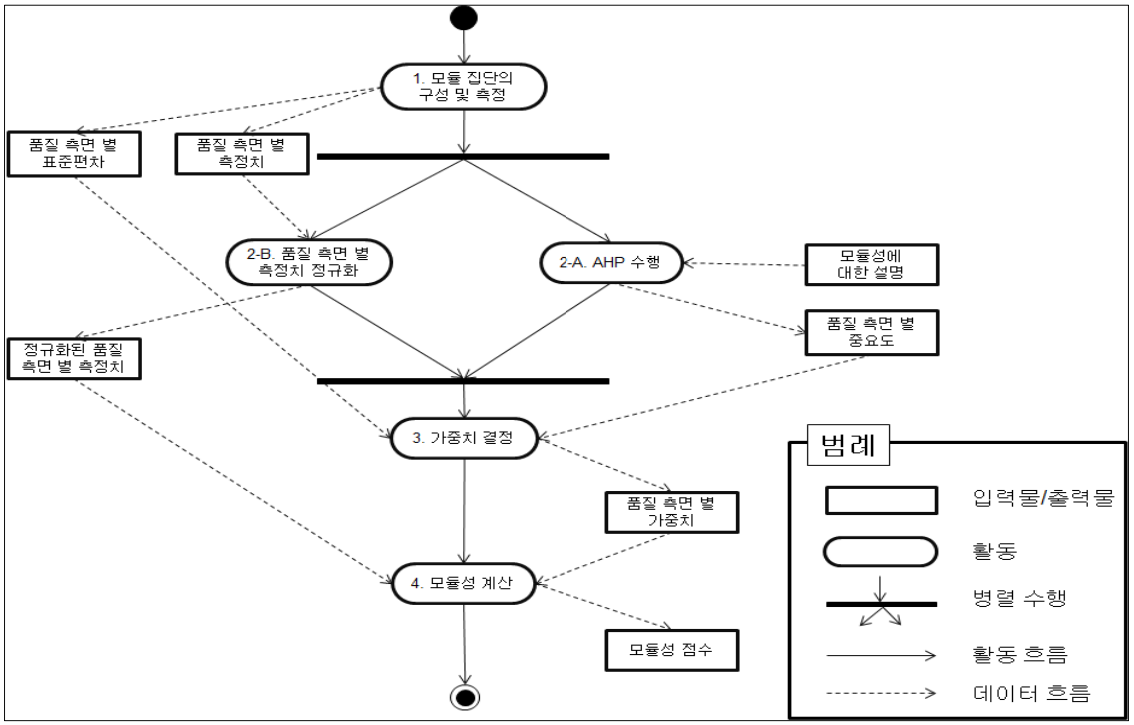
모듈을 구성하는 각 클래스(파일)의 라인 수가 많으면 그만큼 많은 기능들이 한 클래스에 집적되어 유지보수성, 재사용성 등 모듈 전체의 코드 품질을 저하시킨다. 또한, 모듈 내에 동일한 기능을 수행하는 코드가 중복으로 구현되어 있으면 코드 수정 시, 잇따라 수정되어야 할 코드의 미수정으로 인한 결함 발생률을 높인다. 따라서 코드 품질을 결정하는 품질 측면으로 평균 클래스(파일) 라인 수와 중복코드 비율을 선정한다.

<표 2>는 각 품질 속성을 구성하는 품질 측면과 측정 지표를 나타낸다. 품질 측면은 해당 속성에 대해 긍정적인 상관관계를 가질 수도 있고 (+로 표시)부정적인 상관관계를 가질 수도 있다(-로 표시).

모듈은 기본적으로 C++과 Java와 같은 객체지향 언어에서는 클래스가 될 수 있고 C와 같은 절차지향 언어에서는 파일이 될 수 있다. 클래스의 개념이 없는 C언어에서는 상속성, 다형성, 추상성의 측정결과가 0이기 때문에 C++, Java에 비해 모듈성이 떨어진다. 이는 C언어가 지향하는 바는 객체지향과 다르며, 모듈성과 거리가 멀기 때문이다.

<표 2> 모듈성의 구성 품질 속성, 품질 측면 및 측정 지표

품질 속성	품질 측면 (모듈성과의 상관관계)	측정 지표
캡슐화	정보 은닉(+)	$\frac{1-public\text{맴버변수수}}{\text{맴버변수수}}$
맞춤성	결합도(-)	외부 모듈과의 함수 참조 수
	상속성(+)	$\frac{\text{상속하는함수수}}{\text{접근가능한함수수}}$
	다형성(+)	가상함수의 개수
이해 용이성	추상성(-)	조상 클래스 수
	응집도(+)	CAMC(Cohesion Among Methods in a Class)[24]
	복잡도(-)	$\frac{\text{순환복잡도}}{\text{총함수수}}$
	평균 함수 (메소드) 라인 수(-)	$\frac{\text{총라인수}}{\text{총함수수}}$
코드 품질	평균 클래스 (파일) 라인 수(-)	$\frac{\text{총라인수}}{\text{클래스(파일)수}}$
	중복코드 비율(-)	$\frac{\text{중복코드라인수}}{\text{총라인수}}$



[그림 2] 모듈성 계산식 도출 절차

3.3 모듈성 계산식의 도출

모듈성을 측정하기 위해서 다음 두 가지 사항을 고려할 필요가 있다. 첫째, 각 품질 측면은 모듈성에 대해 중요한 정도가 다르다. 그 이유는 각 품질 측면이 모듈성에 미치는 영향력이 다르기 때문이다. 따라서 모듈성을 산출할때, 각 품질 측면은 중요도에 따라 다른 비중으로 합산될 필요가 있다. 이를 위해, 각 품질 측면의 가중치를 결정하는 활동이 필요하다. 둘째, 각 품질 측면 측정값을 합산하여 모듈성 점수를 계산하기 위해서는 각 품질 측면의 측정 단위가 동일해야 한다.

각 품질 측면의 가중치를 결정하기 위해서 AHP를 통해 전문가들의 의견을 수집한다. 이를 활용하여 결정된 가중치와 모듈의 품질 측면 측정값를 조합하여 모듈성을 측정할 수 있다. 각 품질 측면의 측정 단위 차이를 고려하기 위해서 통계적 방법을 적용한다.

[그림 2]는 모듈성의 품질 측면 별 가중치를 결정하여 모듈성을 측정하는 방법을 나타낸 활동도이다. 총 다섯 가지 활동으로 나뉘고 각 활동은 입력물 또는 산출물을 가진다. 본 소절에서는 각 활동에 대해 설명한다.

3.3.1 모듈 집단의 구성 및 측정

활동 1은 모듈성을 측정할 모듈들을 수집하고 각 모듈의 품질 측면들을 측정 지표를 이용하여 측정하는 활동이다. 품질 측면 별 측정값들은 모듈성을 산출하기 위해 사용된다.

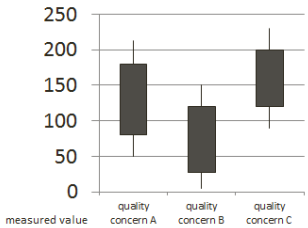
3.3.2 품질 측면 별 측정치 정규화

모듈성을 산출하기 위해 품질 측면 별 측정값들을 바로 사용할 수 없다. 그 이유는 같은 측정값 수치에 대해서 같은 수준을 나타내지 않기 때문이다. 예를 들어, [그림 3]은 여러 모듈에 대한 품질 측면 A, B, C의 측정값 분포를 나타낸다. 이들은

측정값 100에 대해, A와 C품질 측면은 낮은 수준에 해당하지만 B품질 측면은 높은 수준에 해당한다. 높은 수준에 해당하는 측정값에 높은 점수를 부여하고 낮은 수준에 해당하는 측정값에 낮은 점수를 부여하기 위해서는 측정값을 수준에 맞는 점수로 변환할 필요가 있다. 이를 위해, 활동 2-A에서 측정값의 정규화를 수행한다.

정규화는 식(1)을 사용하여 수행한다. 각 품질 측면 측정값(X_i)의 점수(X_i')는 평균($\overline{X}$)을 감산하여 수준을 동일하게 조정할 수 있다. 예를 들어, A 품질 측면 측정값이 1, 2, 3이고 B 품질 측면 측정값이 1001, 1002, 1003이라고 할 때, B 품질 측면 측정값이 A품질 측면 측정값보다 큰 경향을 띈다. 이를 식(1)을 통해 정규화하여 1과 1001에는 -1점, 2와 1002에는 0점, 3과 1003에는 1점을 부여할 수 있다.

(1) $X_i' = X_i - \overline{X}$



[그림 3] 정규화되지 않은 측정값의 분포

3.3.3 AHP 수행

모듈성을 구성하는 각 품질 측면 별 중요도를 결정하기 위해, 여러 전문가들이 AHP를 통해 쌍대 비교를 수행하여 모듈성에 대해 어떤 품질 측면이 얼마나 더 중요한지 판단한다. AHP를 수행하는 전문가들은 먼저 모듈성에 대한 기반 지식을 숙지해야 한다. 그 이유는 모듈성의 각 품질 측면들이 선정된 이유와 측정 방법을 이해해야만 각 품질 측면이 모듈성을 결정하는데 얼마나 중요한지 명확히 판단할 수 있기 때문이다.

각 품질 측면이 $C_1 \cdots C_n$이고 C_i가 C_j보다 중요한 정도를 V_{ij}라고 할 때, 전문가들은 <표 3>을 참고하여 <표 4>를 작성할 수 있다.

<표 3> 상대적 중요도에 대한 값

품질 측면 C_i에 대한 품질 측면 C_j의 상대적 중요도	값
매우 더 중요	5
더 중요	3
동일	1
덜 중요	1/3
매우 덜 중요	1/5

<표 4> 쌍대 비교표

i \ j	C_1	C_2	…	C_n
C_1	1	V_{12}	…	V_{1n}
C_2	1/ V_{12}	1	…	V_{2n}
…	…	…	…	...
C_n	1/ V_{1n}	1/ V_{2n}	…	1

쌍대 비교는 전문가의 주관적 판단에 따라 수행되므로 <표 4>가 일관성 있게 작성되었는지 평가하기 위해 일관성 검사를 수행한다. 일관성은 일관성 지수(Consistency Index: CI)와 일관성 비율(Consistency Ratio: CR)을 계산하여 평가할 수 있다. 일관성 지수는 쌍대 비교의 대상이 되는 품질 측면의 개수를 n, 최대 고유벡터(eigenvector)를 $\lambda_{\max}$라 할 때, 식(2)를 사용하여 계산할 수 있다.

(2) $CI = \frac{\lambda_{\max} - n}{n-1}$

일관성 비율은 일관성 지수와 확률 지수(Random Index: RI)를 사용하여 식(3)을 통해 계산할 수 있다. 일반적으로 일관성 비율이 0.1이하일 경우, 논리적으로 일관성을 유지한 것으로 판단한다[25]. 만일, 전문가가 평가한 쌍대 비교표의 일관성 비율이 0.1을 초과한다면 해당 쌍대 비교표는 일관적으로 판단되지 않은 것이므로 쌍대 비교를 재수행할 필요가 있다.

(3) $CR = \frac{CI}{RI}$

일관성 검사를 통해 일관성이 검증된 전문가들의 평가자료는 기하평균법(geometric average)을 사용하여 종합할 수 있다. 기하평균법은 상대적인 비율값들의 평균치를 계산할 때 사용되는데, 쌍대비교는 두 품질 측면간의 상대적인 중요도 비율을 평가하는 방법이기 때문에 기하평균법을 사용하는 것이 적합하다.

종합된 쌍대 비교표에 식(4)를 이용하여 각 품질 측면(X_i)의 기하평균(G_i)을 계산하고 식(5)을 사용하여 각 품질 측면의 중요도(AHP_{X_i})를 산출한다.

(4) $G_i = \left(\sum_{j=1}^{n} V_{ij} \right)^{\frac{1}{n}}$

(5) $AHP_{X_i} = \frac{G_i}{\sum_{k=1}^{n} G_k}$

3.3.4 가중치 결정

각 품질 측면의 가중치는 앞서 결정한 품질 측면 별 중요도를 이용할 수 있다. 그러나 각 품질 측면은 측정 단위가 다르기 때문에 동일한 수치에 대한 크기가 다르다. 예를 들어, 코드 라인 수에서 1이라는 수치는 크기가 작지만 중복코드 비율에서 1이라는 수치는 굉장히 큰 수치이다. 따라서 단위 차이 문제를 해결하기 위해 모듈 집단의 표준편차를 활용한다. 표준편차는 표본의 측정값이 평균으로부터 얼마나 퍼져있는지 나타낸다. 이는 단위 차이에 따라 큰 영향을 받는다. 이를 해결하기 위해, 각 수치들을 표준편차로 나누어 측정 단위의 영향을 배제시킬 수 있다. 예를 들어, 다음과 같이 집합 A, B, C가 있을 때,

A(cm)={0, 2, 2, 5, 7, 7, 7, 10, 10, 10}
B(mm)={0, 20, 20, 50, 70, 70, 70, 100, 100, 100}
C(kg)={0, 0, 5, 5, 5, 5, 10, 10, 10, 10}

이들을 구성하는 측정값들은 단위가 다르기 때문에 서로 비교할 수 없다. 물론, 집합 A와 B는 실질적으로 크기가 같은 측정값들로 구성되지만 수치상으로는 다르다. 이들을 표준편차로 나누면 다음과 같다.

A′={0, $\frac{2}{\sqrt{12}}, \frac{2}{\sqrt{12}}, \frac{5}{\sqrt{12}}, \frac{7}{\sqrt{12}}, \frac{7}{\sqrt{12}}, \frac{7}{\sqrt{12}}, \frac{10}{\sqrt{12}}, \frac{10}{\sqrt{12}}, \frac{10}{\sqrt{12}}$}
B′={0, $\frac{2}{\sqrt{12}}, \frac{2}{\sqrt{12}}, \frac{5}{\sqrt{12}}, \frac{7}{\sqrt{12}}, \frac{7}{\sqrt{12}}, \frac{7}{\sqrt{12}}, \frac{10}{\sqrt{12}}, \frac{10}{\sqrt{12}}, \frac{10}{\sqrt{12}}$}
C′={0, 0, $\frac{5}{\sqrt{12}}, \frac{5}{\sqrt{12}}, \frac{5}{\sqrt{12}}, \frac{5}{\sqrt{12}}, \frac{10}{\sqrt{12}}, \frac{10}{\sqrt{12}}, \frac{10}{\sqrt{12}}, \frac{10}{\sqrt{12}}$}

집합 A′, B′, C′는 측정 단위에 의한 측정값의 차이를 고려한 결과이다. 따라서 A′와 B′는 같은 측정값들로 구성되어 있다고 볼 수 있다. 마찬가지로, B′와 C′ 또한 단위 차이가 고려되어 수치상으로 서로 비교할 수 있다.

결과적으로, 각 품질 측면의 모듈성에 대한 중요도와 측정단위 차이를 고려한 가중치는 식(6)과 같다.

(6) $\omega_i = \frac{AHP_{X_i}}{\sigma_{X_i}}$

3.3.5 모듈성 계산

여러 가지 품질 측면들을 기준으로 모듈의 모듈성을 결정하는 문제는 다기준 의사결정(Multi-Criteria Decision Making) 문제이다. 다기준 의사결정 문제를 해결하기 위한 간단한 방법 중 하나는 가중합 모델(Weighted-Sum Model)을 이용하는 방법이다. 그 이유는 각 기준들의 점수와 가중치를 단순히 곱의 합으로 쉽게 계산하여 의사결정을 수행할 수 있기 때문이다. n 개의 품질 측면 별 가중치(ω_i)와 측정값(X_i)을 이용하여 모듈성 점수(M)를 산출하기 위한 가중합 모델은 식(7)과 같다.

(7) $M = \sum_{i=1}^{n} \omega_i X_i$

가중합 모델에서 앞서 유도한 가중치와 정규화된 측정값을 적용하기 위해 식(7)에 식(1)과 식(6)을 대입하면 식(8)과 같다. 식(8)을 이용하여 각 품질 측면의 중요도와 측정 단위 차이가 고려된 모듈성을 산출할 수 있다.

(8) $M = \sum_{i=1}^{n} \frac{AHP_{X_i}}{\sigma_{X_i}} \cdot (X_i - \overline{X})$

3.4 검증 방법

모듈성 산출 결과가 모듈의 품질을 잘 반영하는지 검증하기 위해, 제안 방법으로 산출한 모듈성과 전문가들이 직접 판단한 모듈성의 일치도를 확인할 수 있다. 그러나 전문가들이 수집된 모든 모듈의 내부 코드를 분석하여 모듈성을 평가하는 것은 상당히 복잡하고 어렵다. 따라서 수집된 n개의 모듈 중 m($\leq$ n)개를 임의로 선정하고 제안 방법으로 결정한 m개 모듈의 순위와 전문가들이 직접 판단한 m개 모듈의 순위를 비교한다. 두 방법의 순위가 얼마나 밀접한 관계를 가지는지 측정하기 위해 스피어만 순위상관계수(Spearman's rank-order correlation coefficient)(ρ)를 사용할 수 있다. 스피어만 순위상관계수는 두 변수 사이의 상관된 정도를 변수값의 순위를 이용하여 분석하는 방법으로, 각 변수값에 대한 순위 차이(d_i)를 이용하여 식(9)을 통해 계산할 수 있다. 이는 -1과 1사이의 값을 가지는데, 두 변수의 순위가 완전히 일치하면 1, 두 변수가 완전히 반대이면 -1이 된다.

(9) $\rho = 1 - \frac{6\sum d_i^2}{n(n^2-1)}$

<표 5> 스피어만 상관계수 임계치

쌍 개수 \ 유의수준	10%	5%	1%
4	1.0000		
5	0.9000	1.0000	
6	0.8286	0.8857	1.0000
7	0.7143	0.7857	0.9286
8	0.6429	0.7381	0.8810
9	0.6000	0.7000	0.8333
10	0.5636	0.6485	0.7939
11	0.5364	0.6182	0.7545
12	0.5035	0.5874	0.7273
…	…	…	…

스피어만 순위상관계수를 이용하여 두 방법으로 결정한 순위가 서로 관련이 있다는 것을 보이기 위해 유의성 검정(significance test)[26]을 수행할 수 있다. 이를 통해, 산출된 순위상관계수가 '두 방법으로 결정한 순위가 관련이 없다'라는 귀무가설의 임계치를 벗어난다는 사실을 보임으로써 대립가설을 채택할 수 있다. 임계치는 쌍의 개수와 신뢰수준 혹은 유의수준에 따라 결정된다. 일반적으로 90%, 95%, 99%의 신뢰수준 혹은 10%, 5%, 1%의 유의수준을 사용하는 것이 보통이다. <표 5>는 쌍의 개수와 신뢰수준 별 임계치을 나타낸 표이다.

각 전문가에 대해, 두 방법의 순위상관계수가 임계치보다 높으면 귀무가설은 기각되고 두 방법으로 결정한 순위는 상관성이 있다고 판단한다.

4. 오픈소스를 이용한 사례 연구

본 절에서는 오픈소스 소프트웨어를 대상으로 제안 방법을 적용한 사례 연구를 소개한다. 먼저, 4.1절에서는 모듈성을 측정하기 위해 선정된 모듈을 제시하고 그들의 각 품질 측면을 측정한다. 그리고 4.2절에서는 품질 측면 별 가중치 결정을 위한 AHP를 수행 과정을 보이고, 4.2절에서는 각 모듈들의 품질 측면 측정값과 AHP 수행 결과를 이용하여 각 모듈의 모듈성을 측정한다. 마지막으로, 4.4절에서는 제안 방법을 통한 모듈성이 올바로 측정되는지 검증한다.

<표 6> 선정된 오픈소스 소프트웨어

언어	오픈소스	선정 모듈 개수
C	bash, gcc, php, gimp, chrome, linux	33
C++	Filezilla, inkscape, webkit, gwenview, libxml++, 7zip	33
Java	Tomcat, hibernate, spring, junit, Log4j	34
합계	17개	100

<표 7> 종합된 전문가들의 AHP수행 결과

	정보 은닉	결합도	상속성	다형성	추상성	응집도	복잡도	평균 함수 라인 수	평균 클래스 라인 수	중복코드 비율
정보 은닉	1.00	0.47	1.34	1.41	1.20	0.54	1.05	1.90	1.72	1.19
결합도	2.12	1.00	3.30	3.13	2.70	1.25	1.84	3.91	3.11	2.42
상속성	0.75	0.30	1.00	1.16	0.93	0.51	0.90	1.54	1.44	0.79
다형성	0.71	0.32	0.86	1.00	0.84	0.36	0.74	1.57	1.48	0.85
추상성	0.84	0.37	1.07	1.20	1.00	0.42	0.72	1.46	1.24	0.76
응집도	1.86	0.80	1.95	2.81	2.39	1.00	1.63	2.50	2.50	2.12
복잡도	0.95	0.54	1.11	1.35	1.38	0.61	1.00	2.12	2.04	1.56
평균 함수 라인 수	0.53	0.26	0.65	0.64	0.68	0.40	0.47	1.00	1.07	0.68
평균 클래스 라인 수	0.58	0.32	0.70	0.68	0.81	0.40	0.49	0.93	1.00	0.85
중복코드 비율	0.84	0.41	1.27	1.17	1.32	0.47	0.64	1.46	1.17	1.00
중요도	**0.099**	**0.209**	**0.078**	**0.071**	**0.076**	**0.167**	**0.106**	**0.054**	**0.058**	**0.082**

<표 8> 오픈소스 모듈의 모듈성 점수와 순위

모듈명	모듈성 점수	순위	모듈명	모듈성 점수	순위	모듈명	모듈성 점수	순위
FileHandler	0.343	24	CControlSocket	-0.210	75	readline	-0.848	95
AsyncFileHandler	0.140	46	CBookmarksDialog	-0.670	92	relocatable	-0.136	68
MethodExpressionImpl	0.321	27	CFileItem	-0.007	58	node	-0.202	72
ELArithmetic	-0.091	64	CFileZillaApp	0.004	56	policy	-0.577	89
ELParser	-1.439	97	CFilterManager	-0.327	82	trusted	-0.460	87
Token	0.351	22	COptionsPage	0.394	18	internal	-0.203	73
Compiler	-0.271	79	CProxySocket	-0.807	94	symtab	0.081	51
Generator	-0.863	96	CEncoder	0.073	52	mkdeps	0.083	50
ParserController	-0.145	69	CThreadInfo	-0.206	74	runtime	-0.243	77
…	…	…	…	…	…	…	…	…

<표 9> 제안 방법을 통한 모듈 순위와 전문가가 평가한 모듈 순위간의 일치성 검사 결과

모듈명	평가 순위											
	제안 방법	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10	평균
stats	4	6	2	5	2	1	1	1	2	2	4	1
trusted	11	9	11	12	11	7	8	11	7	10	10	10.5
history	8	8	8	8	9	12	11	10	5	8	7	8
gimp	12	12	10	11	12	9	9	12	12	12	11	12
CFilterManager	10	10	9	10	10	8	7	4	11	9	9	9
CData	1	1	1	1	4	11	10	8	4	1	2	3
COptionsPage	2	2	4	2	1	3	3	3	1	3	5	2
CEncoder	7	5	5	3	7	6	4	6	9	5	6	6
ServletServerHttpRequest	5	7	6	4	3	4	5	2	3	6	8	5
Event	3	4	3	6	5	2	2	9	8	4	1	4
TestCaseTest	6	3	7	7	6	5	6	7	6	7	3	7
CControlSocket	9	11	12	9	8	10	8	5	10	11	12	10.5
스피어만 상관계수(ρ)		0.90	0.90	0.90	0.92	0.44	0.52	0.43	0.74	0.94	0.84	0.93
유의성 검정(ρ>0.51)		✓	✓	✓	✓	X	✓	X	✓	✓	✓	✓

4.1 오픈소스 모듈의 선정 및 측정

오픈소스는 마이크로소프트에서 설립한 Ohloh 사에서 제공하는 오픈소스 목록을 사용자 수로 내림차순으로 정렬하여 <표 6>과 같이 선정하였고 이들로부터 100개의 모듈을 임의로 선정하였다. 모듈은 정의에 따라, 관련된 기능들의 집합으로 구성되어야 한다. 따라서 C언어로 구현된 오픈소스에서는 파일 단위로 선정하고 C++과 Java로 구현된 오픈소스에서는 클래스 단위로 선정하였다. 선정된 100개의 모듈들은 <표 2>의 측정 지표를 사용하여 측정되었다.

4.2 AHP 수행

모듈성의 품질 측면 별 중요도를 결정하기 위해 10명의 전문가들을 대상으로 AHP를 수행하였다. 전문가는 소프트웨어 공학을 전공하고 있는 석사과정 학생 3명과 박사과정 학생 3명, 현업 실무자 4명으로 구성되고, 각 전문가는 5년 이상의 개발 경력 또는 소프트웨어 공학 연구 경력을 가진다. 그리고 본 연구를 함께 진행하면서 모듈성에 관한 기반 지식을 상세히 숙지하였다.

각 전문가가 작성한 쌍대 비교표는 모두 일관성 비율이 0.1이하로 측정되었다. 따라서 10개의 쌍대 비교표는 논리적으로 일관성을 유지한다고 판단할 수 있다. <표 7>은 10명의 전문가들의 AHP수행 결과를 종합하고 식(4)와 식(5)를 사용하여 품질 측면 별 중요도를 산출한 결과이다. 여러 가지 품질 측면들 중, 모듈성을 결정하는데 결합도, 응집도, 복잡도 순으로 중요하다는 사실을 알 수 있다.

4.3 모듈성 계산

모듈의 품질 측면 측정값들과 품질 측면 별 중요도를 이용하여 식(8)을 통해 모듈성을 계산할 수 있다. <표 8>은 선정된 모듈의 모듈성 산출 결과와 순위를 나타낸다. 이를 활용하여, 선정된 모듈 내에서 각 모듈의 모듈성 수준을 판단 할 수 있다. 예를 들어, <표 8>의 ELParser의 모듈성 순위는 97위이다. 모듈성 순위가 낮게 산출된 이유는 모듈을 이루는 함수들간의 참조 수가 약 900개로 매우 많기 때문에 수정하기 어렵고, 약 2,000줄의 매우 큰 모듈이면서 그 중 68%의 중복 코드를 지니기 때문에 잠재적 결함 발생률이 매우 높다. 따라서 ELParser는 모듈성 수준이 낮게 평가된다.

측정 결과를 분석한 결과, C로 구현된 모듈보다 C++과 Java로 구현된 모듈의 모듈성 수준이 더 높게 나타나는 경향을 보였다. 이는 객체지향 언어와는 달리, C언어의 지향점이 모듈성과 거리가 있기 때문이다. C언어와 같은 절차지향 언어에 클래스의 개념을 새롭게 추가하여 재사용성과 확장성 등을 높인 언어가 객체지향 언어이다. 따라서 객체지향 언어는 기본적인 모듈성을 갖춘 언어라고 말할 수 있다.

4.4 검증

제안 방법으로 모듈성을 계산한 결과가 모듈의 품질을 잘 반영하는지 검증하기 위해, 본 논문에서는 12개의 모듈을 대상으로 제안 방법을 통해 결정한 순위가 전문가들이 직접 결정한 순위와 높은 일치성을 지닌다는 것을 보인다. 일치성을 평가하기 위해 100개 모듈에서 임의로 선정한 12개 모듈을 대상으로 식(9)를 사용하여 스피어만 순위상관계수를 계산하고 유의성 검정을 수행하였다(<표 9>). 일치성 검사를 위해서 12개의 표본에 대한 5%의 유의수준으로 단측검정(one-tailed test)을 수행하였다. 그리고 임계치(0.5035) 이상의 상관계수를 지니는 전문가를 '√'로 표시하였고, 그렇지 않은 전문가를 'X'로 표시 하였다. 총 10명의 전문가 중, 8명의 전문가가 임계치이상의 상관계수를 지닌다는 것을 알 수 있다. 이는 제안 방법으로 결정한 모듈성이 모듈의 품질을 상당히 잘 반영하고 있다는 것을 의미한다.

5. 결론

본 논문은 소프트웨어 모듈성을 정량적으로 측정하는 방법을 제안하였다. 먼저, 모듈과 모듈성을 정의하고 모듈성을 구성하는 품질 속성과 품질 측면을 정의하였다. 그리고 모듈성을 산출하기 위한 계산식을 도출하고 이를 검증하는 방법을 제시하였다. 사례 연구에서는 오픈소스 소프트웨어 모듈들의 모듈성을 산출하였고, 유의성 검정을 통해 제안 방법의 효용성을 검증하였다.

본 논문의 제안 방법이 앞으로 현업에서 모듈의 활용 방향을 결정하는데 도움을 줄 수 있을 것이다. 또한, 제안 방법의 도구화를 통해 모듈성 평가자들의 노력을 줄여줄 수 있다.

향후 연구로는 각 모듈의 모듈성 수준을 효율적으로 높이기 위해 먼저 개선해야 할 품질 측면들을 선정하는 연구와 각 품질 측면을 개선하기 위해 사용될 수 있는 리펙토링 기법을 제시하는 연구를 수행할 예정이다.

참 고 문 헌

[1] Rosqvist, T., Koskela, M., and Harju, H. “Software Quality Evaluation Based on Expert Judgement.” Software Quality Journal, Vol. 11, No. 1, pp. 39-55, 2003.

[2] Bass, L., Clements, P., and Kazman, R. Software Architecture in Practice, 3rd Edition, Addison-Wesley, 63 page, 2012.

[3] Barbacci, M., Klein, M. H., Longstaff, T. A., and Weinstock, C. B. Quality Attributes. Technical Report, CMU/SEI-95-TR-021, December 1995.

[4] Bangare, S. L., Khare, A. R., and Bangare, P. S. “Measuring the Quality of Object Oriented Software Modularization.” International Journal on Computer Science and Engineering (IJCSE), Vol. 3, No. 1, pp. 445-450, 2011.

[5] Rosqvist, T., Koskela, M., and Harju, H. “Software Quality Evaluation Based on Expert Judgement.” Software Quality Journal, Vol. 11, No. 1, pp. 39-55, 2003.

[6] Sarkar, S., Kak, A. C., and Nagaraja, N. S. “Metrics for analyzing module interactions in large software systems.” Proceedings of the 12th Asia-Pacific Software Engineering Conference (APSEC), pp. 264-271, 2005.

[7] Washizaki, H., Yamamoto, H., & Fukazawa, Y. “A metrics suite for measuring reusability of software components.” Proceedings of the Ninth International Software Metrics Symposium (METRICS), pp. 211-223, 2003.

[8] Sant’Anna, C., Figueiredo, E., Garcia, A., & Lucena, C. J. “On the modularity of software architectures: A concern-driven measurement framework.” Proceedings of the European Conference on Software Architecture (ECSA), Madrid, pp. 207-224, 2007.

[9] Sarkar, S., Rama, G. M., and Kak, A. C. “API-based and information-theoretic metrics for measuring the quality of software modularization.” IEEE Transactions on Software Engineering, Vol. 33, No. 1, pp. 14-32, 2007.

[10] Sarkar, S., Kak, A. C., and Rama, G. M. "Metrics for measuring the quality of modularization of large-scale object-oriented software." IEEE Transactions on Software Engineering, Vol. 34, No. 5, pp. 700-720, 2008.

[11] Li, W., Henry, S. "Object-oriented metrics that predict maintainability." Journal of systems and software, Vol. 23, No. 2, pp. 111-122, 1993.

[12] Bansiya, J., Davis, C. G. "A hierarchical model for object-oriented design quality assessment." IEEE Transactions on Software Engineering, Vol. 28, No. 1, pp. 4-17, 2002.

[13] Subramanian, N., Chung, L. "Metrics for software adaptability." Proceedings of International Concerence on Software Quality Management (SQM), pp. 95-108, 2001.

[14] Liu, X., Wang, Q. "Study on application of a quantitative evaluation approach for software architecture adaptability." Proceedings of the Fifth International Conference on Quality Software (QSIC). pp. 265-272, 2005.

[15] Clements, P., Garlan, D., Bass, L., Stafford, J., Nord, R., Ivers, J., and Little, R. Documenting software architectures: views and beyond, 2nd Edition, Pearson Education, 56 page, 2010.

[15] Briand, L. C., Morasca, S., and Basili, V. R. "Property-based software engineering measurement." Transactions on Software Engineering, Vol. 22, No. 1, pp. 68-86, 1996.

[16] Rozanski, N., Woods, E. Software systems architecture: working with stakeholders using viewpoints and perspectives, Addison-Wesley, 21-22 pages, 2011.

[17] Hofmeister, C. Applied software architecture. Addison-Wesley. 106-110 pages, 2000.

[18] Bloch, J. Effective java, Addison-Wesley, 2008.

[19] Pressman, R. S., and Ince, D. Software engineering: a practitioner's approach, 7th Edition, McGraw-Hill, 400-401 pages, 1992.

[20] Luer, C. "Assessing Module Reusability." Proceedings of the First International Workshop on Assessment of Contemporary Modularization Techniques (ACoM'07), IEEE Computer Society, p. 7, 2007.

[21] Parnas, D. L. "On the criteria to be used in decomposing systems into modules." Communications of the ACM, Vol. 15, No. 12, pp. 1053-1058, 1972.

[22] Chidamber, S. R., & Kemerer, C. F. "A metrics suite for object oriented design", IEEE Transactions on Software Engineering, Vol. 20, No. 6, pp. 476-493, 1994.

[23] Briand, L. C., Wust, J., & Lounis, H. "Using coupling measurement for impact analysis in object-oriented systems". International Conference on Software Maintenance, pp. 475-482, 1999.

[24] Bansiya J., Etzkorn L., Davis C. and Li W. "A Class Cohesion Metric For Object-Oriented Designs", The Journal of Object-Oriented Programming, Vol. 11, No. 8, pp. 47-52, 1999.

[25] Saaty, R. W. "The analytic hierarchy process—what it is and how it is used." Mathematical Modelling, Vol. 9, No. 3, pp. 161-176, 1987.

[26] Zar, J. H. "Significance testing of the Spearman rank correlation coefficient". Journal of the American Statistical Association, Vol. 67, No. 339, pp. 578-580, 1972.

저자 소개

정 필 수

2012년 충남대학교 컴퓨터공학과 졸업(학사)
2012~현재 한국과학기술원 전산학과 석사과정
관심분야는 소프트웨어 아키텍처, 소프트웨어 테스팅

안 종 선

2008년 건국대학교 전자공학과 졸업(학사)
2011년 카이스트 우주탐사공학과 졸업(석사)
2012년~현재 카이스트 전산학과 박사과정
관심분야는 소프트웨어 품질, 소프트웨어 아키텍처

박 태 현

2010년 KAIST 전산학과 졸업(학사)
2012년 KAIST-CMU 소프트웨어공학 프로그램 졸업 (석사)
2012년~현재 KAIST 전산학과 박사과정
관심분야는 소프트웨어 아키텍처, 소프트웨어 제품라인 공학, 소프트웨어 테스팅

강 성 원

1982년 서울대학교 사회과학대학 졸업(학사)
1989년 University of Iowa 전산학 졸업(석사)
1992년 University of Iowa 전산학 졸업(박사)
1993년~2001년 한국통신(KT) 선임연구원
1995년~1996년 미국 국립표준기술연구소(NIST) 객원연구원
2001년~2005년 한국정보통신대학교 조교수
2003년~현재 Carnegie-Mellon University MSE프로그램 겸임교수
2005년~2009년 한국정보통신대학교 부교수
2009년~현재 KAIST 전산학과 부교수

은 나 래

2004년 이화여대 컴퓨터공학과 졸업(학사)
2006년 이화여대 컴퓨터공학과 졸업(석사)
2006년~현재 LG전자 SW역량강화센터

고 상 원

2009년 한양대 컴퓨터공학과 졸업(석사)
2009년~2012년 롯데정보통신
2012년~현재 LG전자 SW역량강화센터