ISSN 2586-1816
융합정보논문지 제7권 제5호 pp. 111-116, 2017 DOI : https://doi.org/10.22156/CS4SMB.2017.7.5.111

아키텍처 자산의 복잡도 측정에 관한 연구

최한용*
신한대학교 IT융합공학부

A Study on the Complexity Measurement of Architecture Assets

Han-Yong Choi*
Division of IT Convergence Engineering, Shinhan University

요 약 자산의 복잡성을 측정하기 위해 프로그램의 논리적인 복잡도의 측정을 제공하는 척도를 베이스로 하여 각 자산의 특징 값을 표현하고 있는지 평가하는 방법을 사용 한다. 본 연구에서는 소프트웨어 컴포넌트를 기본자산으로 구성하여 표준화된 설계모형을 확보하고, 이를 기반으로 재사용 가능한 확장된 자산을 설계할 경우 자산의 복잡도를 측정하기 위한 방안을 제시하고자 한다. 그러나 우리가 제안하는 자산관리 시스템의 각 자산은 두 가지 영역의 자산을 합성한 복합자산으로 구성되어 있으므로 이 방법만으로는 정확한 측정을 하기 어렵다. 따라서 아키텍처 하부에 저장된 기본 자산의 특성 값을 반영하여야 전체적인 자산의 복잡성을 측정가능하다. 따라서 응집력에 반비례하고 자산연관도안의 각 자산에 대한 연관값의 누적합에 비례하는 합성자산의 복잡성을 측정 가능하다.

키워드 : 자산, 재사용, 복잡도, 아키텍처, 컴포넌트

Abstract In this paper, we propose a method to measure the complexity of assets when a software component is constructed as a basic asset, a standardized design model is acquired, and a reusable extended asset is designed based on the standardized design model. However, each asset of our proposed asset management system consists of composite assets that combine assets of two domains. So this method can not make accurate measurements. Therefore, the complexity of the overall asset can be measured by reflecting the property value of the basic asset stored under the architecture. In conclusion, it is possible to measure the composite-complexity of a composed asset that is inversely proportional to cohesion and proportional to the cumulative sum of the associated values of each asset in the asset-related design.

Key Words : Asset, Reuse, Complexity, Architecture, Component

1. 서론

소프트웨어 산업에서 생산성에 대한 연구가 장기간 이루어져 왔으며, 생산의 효율성을 향상시키기 위한 자동화 방법을 다양하게 연구하고 있다. 이를 해결하기 위한 효과적인 방법으로 시스템 개발 과정에서 잘 추상화된 설계정보의 활용에 대한 요구가 증가하였다. 또한 다양한 요구사항의 변화는 기술적으로 흡수되기를 원하고 있으며, 소프트웨어의 생산효율을 증가시키기 위한 방법론과 표준에 대한 연구와 정형화된 소프트웨 생산을 위한 재사용에 어려움을 겪고 있다[1,2]. 본 연구에서는 소프트웨어 아키텍처 컴포넌트를 기본자산으로 구성하여 표준화된 설계모형을 확보하고, 이를 기반으로 재사용 가능한 확장된 자산을 설계할 경우 자산의 복잡도를 측정하기 위한 방안을 제시하고자 한다. 복잡도 측정을 위해 개발 환경에 독립적이고 표준화된 설계단계에서 도메인 설계정보를 재사용하고 이를 기반으로 합성가능한 부품자산을 재사용하는 경우를 대상으로 한다. 그러나 설

Received 2017-09-29 Revised 2017-10-11 Accepted 2017-10-20 Published 2017-10-31
*Corresponding author : Han-Yong Choi (hychoi@shinhan.ac.kr)

계단계에서는 설계대상이 되는 도메인의 아키텍처와 응용영역에 필요한 최적화된 자산을 필요로 하고 있다. 그 이유는 표준화하여 정의된 설계정보를 도메인에 맞게 정형화하여 설계자들이 재사용할 수 있는 자산을 제공하고 있지 못하며, 제공되는 설계 자산 정보를 합성하여 복합 잔사의 설계정보를 확장할 수 있는 환경을 제공하지 못하고 있다는 것을 의미한다. 그리고 설계단계에서 부품 자산의 재사용을 적용하기 위해 이미 설계된 설계정보를 합성하고 이를 수용할 수 있는 아키텍처 레벨의 재사용을 위해 자산의 최적화 과정에서 복잡도가 증가는 문제를 가질 수 있다.

본 연구에서 사용하려는 자산의 재사용을 위한 아키텍처는 도메인 기반의 플랫폼에서 재사용 가능한 자산 설계를 가능하게 한다. 그리고 기존의 자산정보를 재사용하기 위해 정형화하는 문제점과 플랫폼을 기반으로 소프트웨어를 설계하기 위한 문제를 해결하기 위해 PLE 구조를 적용하였으며, 이때 표준화된 기초 자산과 복합 자산의 복잡도를 측정하는 방법을 연구하고자 한다. 또한 자산 정보의 설계에서는 재사용성에 대한 고려와 조립성을 고려하여 아키텍처 자산을 정제하여야 하며, 그렇게 함으로써 제품 라인을 위한 제품은 재사용 가능한 자산을 기반으로 설계된 자산 컴포넌트로부터 합성이 가능하게 자산을 사용할 수 있다.

따라서 본 연구에서는 설계정보에서 자산을 정제하기 위해 기본 자산으로부터 확장되는 재사용 자산을 설계할 경우 복잡도의 변화를 측정하기 위한 지표를 마련하는 것을 목표로 한다. 이때 각 자산의 최적화 과정에서 자산의 복잡도를 측정하는 방법을 제안하여 플랫폼 기반의 재사용 가능한 자산의 재사용시 복잡도가 주는 영향을 측정하여 복잡도를 감소시킬 수 있는 자산의 재사용 방법을 제안하고자 한다.

2. 기반연구

2.1 Core RAS

OMG의 공개표준 명세서인 RAS를 기반으로 소프트웨어 자산 관리를 위한 명세를 하고 있다. 이 명세의 내용은 재사용이 가능한 프로그램 자산의 아키텍처와 콘텐츠 그리고 그 내용에 대한 설명을 기술한 재사용 절차의 일관성을 위한 명세방법이다. 아키텍처 자산은 자산 명세르 위한 기본 구성요소들을 Core RAS와 같이 모두 포함하고 있으며, Fig. 1은 RAS의 UML 모델을 나타내고 있다[3].

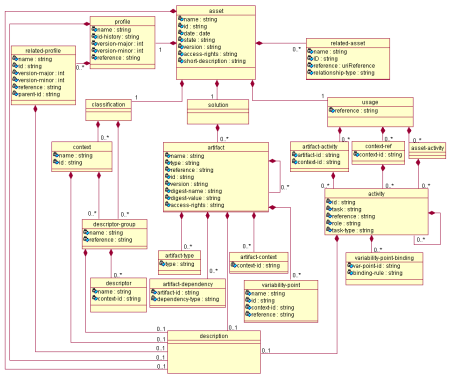

Fig. 1. Core RAS UML Model[3]

그리고 종속적인 형태의 자산들에 추가적 정보를 소개하기 위해 확장된 프로파일 기능을 갖도록 응용자산을 정의한다. Core RAS는 4개의 섹션으로 구성되어 있다[3]. 자산을 분류하기 위한 Classification 섹션은 일련의 명세서들을 열거하며, 자산의 산출물들에 대하여 기술하는 것은 Solution 섹션에서 담당 한다. 자산을 사용하고 커스터마이즈하기 위한 규칙은 Usage 섹션에서, 그리고 다른 자산들과의 관계는 Related assets 섹션에 기술한다[3]. 본 논문에서는 자산의 계층적 분류가 이루어지도록 범주를 설정하고 자산의 행위적 특성에 따라 분류되는 방법을 사용하였다.

아키텍처 자산에 정보를 분석하기 위해 각 자산에 대한 키워드 분석을 하게 되며, 각 자산에 포함되어 있는 Term들을 컴포넌트의 명과 함께 분석한다[4]. 이때 아키텍처에 저장할 자산의 키워드와 자산의 명칭은 Term-asset의 관계가 된다. 그리고 이때 생성된 많은 Term들을 개념범주로 다시 분류하여 Term-ConceptCategory의 관계 매트릭스를 생성한다. 개념범주는 Term과 자산 사이의 매개체 역할을 한다. 여기서 Term-ConceptCategory 매트릭스가 생성되고, 카이제곱 통계량 계산을 통하여 최적의 Term의 수를 구한다. 또한 검색 효율을 위해 Term들은 개념범주와의 매칭 또는 비매칭 계산에 활용되어 Term-Term 사이의 유의어매트릭스를 구성하였다.

2.2 Asset Configuration

여러 개의 산출물로 구성된 자산은 각각 다른 산출물

을 포함할 수 있으며, 다른 산출물들과 관계를 가질 수 있다[3,5]. 이때 산출물은 요구사항 개발이나 설계 및 런타임 문맥에 해당하는 특정 문맥과 관련된다.

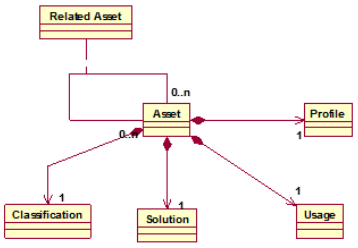

Fig. 2. Asset Configuration[3]

산출물은 산출물의 모델과 각 산출물에 대한 구성요소의 형태로 특수화될 수 있다[3,6]. 모델은 모델, 다이어그램, 명세서로 구성되며, 각각의 형태에 의해 산출물의 구성요소가 내포하는 가변점을 표현하여야 한다. 산출물의 구성요소는 재사용할 경우, 수정이 가해질 수 있는 지점을 나타내기 위한 가변점(Variability point)을 가질 수 있다.

본 논문에서 사용하는 자산은 공개표준 명세서인 RAS를 기반으로 Fig. 2와 같이 명세하였다[3]. 이 명세는 재사용 절차의 일관성을 위하여 재사용 가능한 설계 자산들의 아키텍처(Architecture), 내용 그리고 설명에 대하여 기술하고 있다. 두 영역으로 분리하여 표현된 자산은 다음과 같이 명세하였다. 아키텍처 자산은 Core RAS와 같이 자산 명세에 대한 기본 구성요소들을 모두 포함하고 있다. 그리고 응용자산은 특정 형태의 자산들에 대한 추가적 의미를 소개하기 위해 Core RAS의 확장을 나타내는 프로파일 기능을 갖게 된다. 본 논문에서는 자산의 계층적 분류가 이루어지도록 범주를 설정하고 자산의 행위적 특성에 따라 분류되는 방법으로 제한하여 사용하였다.

2.3 복잡도

소프트웨어공학의 영역 중 소프트웨어에 대한 복잡도는 소프트웨어를 개발하고 운영, 유지보수하기 위한 비용에 관련된 소프트웨어의 특성을 규명하고 분류하거나 측정하기 위한 영역이다[1,7].

소프트웨어의 복잡도는 IEEE(Institute of Electrical and Electronics Engineers) 용어집에 따르면 “시스템이나 시스템 요소들 간의 복잡한 정도로서 데이터 구조의 형태, 중첩된 정도, 조건 분기의 수, 인터페이스 수, 그리고 시스템의 특징 등과 같은 요소를 분석함으로서 결정된다.”라고 정의되어 있으나 보편적으로 프로그램 개발자의 수행 정도에 영향을 주는 소프트웨어의 특성을 나타내는 심리적 복잡도로 정의된다[6]. 복잡도 측정을 위한 요인을 살펴보면 소프트웨어 코드 안의 구성항목인 클래스 수, 한 클래스에서 존재하는 메소드의 수, 메소드 안의 평균 LOC 크기, 클래스 상속 계층 구조의 높이, 클래스당 평균 인스턴스 변수의 수 등에 의해 결정된다[1,8]. 하나의 소프트웨 개발을 위해서, 하나의 클래스는 실세계의 하나의 실체로 나타낼 수 있으므로 다양한 클래스가 하나의 소프트웨어 코드내에 존재하게 되면 코드의 구성과 이해가 어렵고 유지 보수에 어려움을 갖게 된다. 따라서 하나의 모듈안에 클래스 형태의 부품단위는 7±2개 이하가 바람직한 것으로 가이드 되고 있다[8,9]. 하나의 클래스에 많은 종류 메소드나 많은 수의 메소드가 존재하면 클래스 오브젝트의 구성 및 이해가 어렵고 유지보수 에도 많은 어려움이 있다. 그리고 결과적으로 응집력이 낮아지게 되어 클래스의 캡슐화 속성을 악화시킨다. 결국 메소드의 수가 많은 클래스는 재사용의 기능성을 제한하거나 응용영역이 좁아지게 된다. 한 클래스의 응집도가 낮게 되면 클래스는 두 개 또는 그 이상의 서브클래스로 나누어져야 하게 되며 이는 복잡도와도 상관관계를 가질 수 있다. 메소드가 너무 크게 되면 또한 메소드를 유지보수 하는데 비용이 많이 발생하게 된다. 또한 해당 메소드를 포함하고 있는 클래스의 크기가 커지게 되면 복잡도는 증가하고 이를 위한 유지관리가 어려워진다[10].

3. 자산의 복잡도 측정

3.1 기본 자산의 복잡도

아키텍처 자산은 표준화된 정보영역의 설계 값을 표현한 영역과 응용도메인의 설계정보를 표현하는 영역으로 구성되어 있다[11,12]. 자산의 복잡성을 측정하기 위해 프로그램의 논리적인 복잡도의 분량상 측정을 제공하는 척도를 베이스로 하여 각 자산의 특징 값을 표현하고 있는지 평가하는 방법을 제안하고자 한다.

표준 설계정보로 구성된 자산은 기본적인 복잡도의 측정방법인 순환복잡도 측정 식을 이용하여 측정할 수

있다. 복잡도에 대해 계산된 값은 자산의 기본 집합에 독립적인 경로들의 수를 수정한다. 그리고 모든 코드가 최소한은 실행되어 진 것을 검증하기 위해 진행되어야 하는 테스트 수의 상한 값에 대한 경계를 제공한다. 이때 자산 구성의 영역 수는 순환 복잡도와 일치하므로 자산 A에 대한 복잡도 Com(A)는 수식 1과 같이 정의할 수 있다.

$$Com(A) = Ass_{ass} - Ass_{com} + 2P \qquad (1)$$

이와 같이 기본 경로 순환 방식은 그래프 이론에 기본을 두고 있으므로 표준화된 일반적인 자산의 복잡도 측정에 적용하여 유용한 척도를 제공받을 수 있다.

3.2 복합자산의 복잡도

그러나 우리가 제안하는 자산관리 시스템의 각 자산은 두 가지 영역의 자산을 합성한 복합자산으로 구성되어 있으므로 코어 자산에 측정에 적합한 수식 1의 방법만으로는 정확한 측정을 하기 어렵다. 따라서 아키텍처 하부에 저장된 기본 자산의 특성 값을 반영하여야 전체적인 자산의 복잡성을 측정가능하다.

이때 자산의 특성 값은 응집력으로 평가할 수 있으므로 각 기본 자산의 AC 값을 합한 값이다. 그러나 단순히 각 기본 자산의 AC 값을 누적하게 되면 , AC 값은 항상 1 이상이 되기 때문에 모든 기본 자산이 독립되어 있는 자산의 기본 자산 수에 의해 그 값이 다르게 평가된다. 따라서 이런 점을 없애기 위해 수식 2와 같이 자산의 특성 값을 측정하기 위한 자산 응집력을 측정하기 위한 방법을 정의한다.

$$CyCom = \sum_{j=1}^{m} \tau(j)$$
$$\tau(j) = \begin{cases} 0, & AC_j = 1 \\ AC_j, & otherwise \end{cases} \qquad (2)$$
$$m \ : \ number\ of\ assets$$

자산 응집도의 값은 클수록 그 자산의 응집도가 좋다는 것을 의미한다. 그러나 일반적인 복잡도에서 복잡도의 값이 크게 되면 그 자산의 내부정보에 대한 복잡성이 크다는 것을 나타내기 때문에, 이와 같은 서로의 혼동을 막기 위한 방법으로 정의된 복잡도는 정규화하여 사용할 필요가 있다.

이 정규화된 값이 아키텍처 하부에 정의된 일반적인 자산의 복잡성을 나타내고 있다고 볼 수 있다. 그리고 아키텍처 하부의 정의된 자산은 재사용되어 응용도메인에 맞는 합성된 자산을 구성하게 된다. 이때 순환 복잡도를 적용하여 정규화된 값으로 측정된 독립 자산이 합성자산으로 구성될 때 자산간의 연관정도에 의해 복잡도로 측정이 가능하다. 이 두 값을 이용하면 아키텍처 자산으로 재사용되어 얻어진 합성 자산의 복잡도를 측정할 수 있다. 따라서 응집력에 반비례하고 자산연관도의 각 자산의 연관값의 누적합에 비례하는 합성자산의 복잡성을 측정 가능하다.

$$AC_j = \frac{\sum_{i \in I_j} \mu(i)}{|I_j|}$$
AC_i : *set of assets*
I_j : set of composite assets used by M_i assets (3)
$\mu(i)$: number of assets using composite asset i

이와 같이 수식 3에 정의된 자산 연관 척도에 의해 자산내에 있는 각 기본자산과 다른 기본 자산들 간의 연관성을 측정할 수 있다. 이 값은 기본자산이 다른 기본자산과 공유하는 파생 자산이 많으면 그 값이 커지므로, 그 값이 크면 클수록 그 자산은 다른 자산들과 많은 연관성을 가진다고 볼 수 있다. 그리고 어떤 자산이 다른 자산과 공유하는 파생자산이 전혀 없다면 AC 값은 1이 된다. 그 이유는 AC 값이 1이 되기 위해서는 I_j와 분자의 값이 동일해야 하는데, I_j는 그 자산이 사용하는 파생자산의 수이고, 분자는 그 파생자산을 사용하는 자산 수의 합이다. 자산내의 클래스에서 볼 수 있듯이, 다른 자산들과 공유하는 파생 자산 없이 분리되어 있는 자산의 AC값은 1이므로 분리되어 있는 자산들을 찾을 수 있다. 또한 AC 값이 높은 자산은 다른 자산들과 파생 자산에 대한 공유값이 크다는 것을 알 수 있으며, 이것은 자산의 의존성을 증가시키게 된다.

4. 결론

자산을 기반으로 하는 재사용 시스템에서 설계정보자산 정보는 도메인의 속성이나 플랫폼에 독립적이어야 하며, 부품자산으로써 사용하기 위해 재사용성과 조립성의 기본 특성을 만족시켜야 한다.

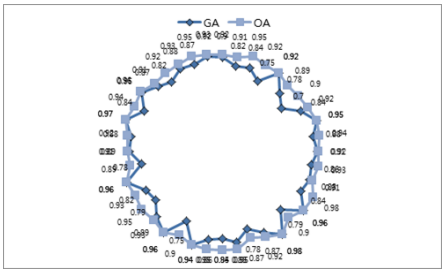

Fig. 3. Accuracy Through Optimization

또한 아키텍처 컴포넌트를 정제하여 최적화한 표준화된 자산을 사용할 수 있으며, 각 부품자산은 설계자의 도메인에 따른 목적에 따라 유연성을 제공하고 독립적으로 합성 가능한 부품이어야 한다. 선행연구에서 자산의 복잡도는 Fig. 3에 제시된 것과 같이 최적화 과정에서 아키텍처 자산의 재사용을 위한 정확도는 증가하였고 이 자산들은 설계정보의 복합성에 대한 평가항목에는 크게 영향을 받지 않은 상황에서 이루어진 결론을 얻은 것이다[13]. 이는 검색 후보자산의 컨텍스트에 의해 재사용을 하게 되므로 자산정보의 최적화 과정에서 복잡도의 변화가 나타나지 않는 것을 알 수 있다. 이것은 최적화에 의해 자산의 재사용 검색효율은 증가하였고, 각 자산의 복잡도에는 영향을 주지 않는다는 결론을 얻었다.

Table 1. Complexity Evaluation

	BM	CM
Independent Asset	O	O
Architecture Asset	O	O
Application Asset	O	O
Composite Asset	O	O
Composite Accuracy	X	O
Composite Association	X	O

Table 1의 결과에서처럼 기본적인 잡도 측정방식(BM)에서는 독립 형태의 자산과 복합 자산의 구분을 하지 않기 때문에 동일한 복잡도로 측정된다. 그리고 이 방식은 복합 자산의 정밀도나 연관도를 측정할 수 없다. 본 논문에서 제안한 복합 자산에 대한 측정방식(CM)은 독립자산의 측정식을 계승 받았고, 복합 자산의 경우 자산의 구성인 기본 자산들 각각의 응집력과 연관성을 반영하여 측정하도록 하였다. 따라서 복합 정밀도와 연관도를 반영하여 복잡도를 측정할 수 있는 모델이다. 코어자산과 응용자산으로 구성된 자산의 복잡도를 측정하기 위해 본 논문에서 제안한 복합 자산의 복잡도 측정방법을 적용할 수 있었다. 코어 자산의 경우 순환복잡도의 특성을 계승하여 측정할 수 있었으며, 이 방법만으로는 복합자산의 측정에 적합하지 않기 때문에 복합자산의 응집력과 자산내부의 연관도를 측정하고 이를 적용한 복합 자산 측정식을 적용하도록 제안하였다. 향후 단일 측정식과 복합 자산 측정식을 프레임워크내의 자산에 적용하여 평과결과로부터 최적화를 위한 과정이 필요하다.

향후 연구과제로 자산의 복잡성을 측정하는 다양한 방법이 각 도메인 영역이나 재사용 방식에 따라 어떠한 영향을 주는지에 대한 평가가 필요할 것으로 판단된다.

ACKNOWLEDGMENTS

본 논문은 2017년도 신한대학교 학술연구비 지원으로 연구되었음.

REFERENCES

[1] R. S. Pressman. (2005). *Software engineering: a practitioner's approach.* London : Palgrave Macmillan.

[2] H. Y. Choi. (2016). MetaData Structure Design of Architecture Asset in DMI. *Journal of IT Convergence Society for SMB, 6(4),* 151-156.
DOI : 10.22156/cs4smb.2016.6.4.151

[3] OMG. (2017). *Reusable Asset Specification OMG Available Specification Version 2.2.* OMG. http://www.omg.org.

[4] K. Pohl, G. Böckle & F. J. van Der Linden. (2005). *Software product line engineering: foundations, principles and techniques.* Berlin : Springer Science & Business Media.

[5] S. Y. Min, S. H. Park & N. H. Lee. (2011). SW Quality of Convergence Product: Char acter istics, Improvement Str ategies and Alternatives. *Journal of IT Convergence Society for SMB, 1(1),* 19-28.

[6] N. I. Altintas, S. Cetin, A. H. Dogru & H. Oguztuzun. (2012). Modeling product line software assets using domain-specific kits. *IEEE transactions on software engineering, 38(6),* 1376-1402.
DOI : 10.1109/tse.2011.109

[7] H. Krahn, B. Rumpe & S. Völkel. (2010). MontiCore: a framework for compositional development of domain specific languages. *International Journal on Software*

Tools for Technology Transfer (STTT), 12(5), 353-372.
DOI : 10.1007/s10009-010-0142-1

[8] S. Moser & V. B. Misic. (1997). Measuring class coupling and cohesion: a formal metamodel approach. *In Software Engineering Conference* (pp. 31-40). USA : IEEE.
DOI : 10.1109/apsec.1997.640159

[9] S. Bernardi, J. Merseguer & D. C. Petriu. (2012). Dependability modeling and analysis of software systems specified with UML. *ACM Computing Surveys, 45(1),* 2.
DOI : 10.1145/2379776.2379778

[10] A. B. Younes, Y. B. Hlaoui & L. J. B. Ayed. (2014). A meta-model transformation from uml activity diagrams to event-b models. *2014 IEEE 38th International (pp. 740-745).* IEEE. USA : IEEE.
DOI : 10.1109/compsacw.2014.119

[11] H. Choi & S. Sim. (2015). A Study on Software Development method based on DMI. *International Conference for Small & Medium Business* (pp. 359-360). Seoul : ICSMB.

[12] F. B. Bryant & A. Satorra. (2012). Principles and practice of scaled difference chi-square testing. *Structural Equation Modeling: A Multidisciplinary Journal, 19(3),* 372-398.
DOI : 10.1080/10705511.2012.687671

[13] H. Choi & S. Sim. (2017). A Study on the Optimization of Architecture Assets in DMI. *Journal of Advanced Research in Dynamical and Control Systems,* 143-149.

저 자 소 개

최 한 용(Han-Yong Choi) [정회원]

- 1993년 2월 : 경희대학교 전자계산공학과 학사
- 1998년 2월 : 경희대학교 전자계산공학과 석사
- 2002년 8월 : 경희대학교 전자계산공학과 박사
- 2004년 3월 ~ 현재 : 신한대학교 IT융합공학부 교수

<관심분야> : 재사용, 플랫폼 디자인, PLE