Journal of Digital Convergence
Vol. 19. No. 8, pp. 219-224, 2021

ISSN 2713-6434 / eISSN 2713-6442
https://doi.org/10.14400/JDC.2021.19.8.219

재구성 가능한 소프트웨어 시스템의 적용

최한용
신한대학교 IT융합공학부 컴퓨터공학전공 교수

The Application of Reconfigurable Software Systems

Hanyong Choi
Professor, Division of IT Convergence Engineering, Shinhan University

요 약 소프트웨어 시장은 다양한 산업의 융합과 함께 적용분야의 경계가 없어지고 융합분야의 제약이 사라졌다. 소프트웨어의 요구사항은 다변화하고 빠른 주기로 소프트웨어 요구사항을 재구성하기를 원하고 있다. 요구사항의 다양한 변화는 기술적으로 수용되어야 하기 때문에 소프트웨어의 생산성에 대한 효율을 높이기 위한 다양한 방법과 표준에 대한 연구, 그리고 이를 위해 소프트웨어를 정형화하여 생산할 수 있는 방법이 필요하다. 본 연구에서는 선행 연구에서 최적화한 자산의 활용을 위해 재구성 가능한 소프트웨어 자산을 적용하였을 때 개발자의 특성과 환경에 따라 자산의 구성에 대한 재사용성과 복잡도가 어떻게 나타나는지 연구하였다. 이때 개발자의 특성에 따라 나타나는 사용성과 자산 구성방법에 따른 복잡도의 변화가 어떻게 나타나는지 측정하였으나 수집 데이터의 한계가 있어 계속적인 데이터 수집으로 측정값의 품질을 확보가 필요하다. 또한 복합 자산의 사용단계에서 컨텍스트 분류의 문제점을 보완하기 위한 지능형 시스템 적용방안이 필요하다.

주제어 : 자산, 최적화, 효율성, 아키텍처, 복잡도

Abstract The convergence of various industries has removed the boundaries of software application fields and reduced the restrictions on convergence fields. Software requirements are diversified and they want to reconfigure software requirements in a fast cycle. Since various changes in requirements have to be accepted technically, research on methodologies and standards to increase the efficiency of software productivity and methods for standardizing and producing software are needed. In this study, we studied how the reusability and complexity of the software asset reconfiguration system appeared according to the developer's characteristics and environment to utilize the assets optimized in previous studies. At this time, we measured how the change in complexity according to the usability and asset composition method that appears according to the developer's characteristics appears, but there is a limit to the collected data, so it is necessary to secure the quality of the measured value through continuous data collection. In addition, an intelligent system application plan is needed to supplement the problem of context classification in the use stage of complex assets.

Key Words : Asset, Optimization, Efficiency, Architecture, Complexity

*이 논문은 2019학년도 신한대학교 교수연구년연수제도에 의하여 연구된 논문
*Corresponding Author : Hanyong Choi(hychoi@shinhan.ac.kr)
Received June 10, 2021 Revised June 28, 2021
Accepted August 20, 2021 Published August 28, 2021

1. 서론

소프트웨어 시장은 적용 분야의 경계가 없어지고 융합 분야의 제약이 사라졌다. 소프트웨어 시장에서는 분야별 다양성 확보를 위한 요구사항은 다변화하고, 빠른 주기의 소프트웨어 재구성을 요구하고 있는 상황이다[1,2]. 요구사항은 빠른 시장의 변화에 따라 다양화되어가고 이를 기술적으로 수용할 수 있어야 하고 또한 빠른 생명주기에 따라 생산성에 대한 효율성을 증가와 소프트웨어 구성을 위한 각 영역의 표준, 그리고 이를 정형화하여 소프트웨어 제품을 생산하기 위한 방법이 필요하다[3]. 또한 CBD를 기반으로 하는 소프트웨어 개발 방법에 근거하여 형상관리의 요구사항을 살펴봐야 한다. 이때 컴포넌트의 버전 관리와 무결성 유지로 일관성 체크와 형상 항목들 사이의 의존성을 관리하는 것으로 구분된다. 자산의 형상관리를 위해서는 컴포넌트간의 의존관리가 필수적인 행위이다[4-6]. 컴포넌트 합성과정에서 컴포넌트의 추가연산이 수행될 때 바이너리 컴포넌트가 직접 전달되지 않기 때문에 일반적으로 컴포넌트 사이의 의존관계 정보까지는 전달되지 않는다. 본 논문에서는 소프트웨어의 기본 구성요소에 해당하는 아키텍처를 대상으로 컴포넌트를 이용하여 기본자산으로 구성하였다. 그리고 구성한 기본 자산을 기반으로 하여 모델을 정형화하거나 표준화하여 재사용을 위한 설계모델을 확보하였다[7,8]. 그리고 자산은 모델 안에서 정형화되어 표현되거나 표준화하여 구성한 후 재사용이 가능한 확장된 자산을 설계하기 위해서 설계 과정에서 각 자산의 사용성에 대한 평가 모델을 구축하기 위한 방법을 선행하였다[9]. 선행 연구에서 사용한 자산의 구조는 RAS를 표준 명세방식으로 이용하였다. 이 방법은 일관성 있는 재사용을 하기 위해 설계된 각각의 자산들에 대하여 아키텍처, 내용 그리고 설명을 포함하여 나타내고 있다. 이때 특성 값이 서로 다르다면 각 자산은 다음과 같이 명세할 수 있다. 표준 영역의 아키텍처는 자산 명세에 대한 기본 구성요소들을 포함하고 있다. 그리고 설계자의 설계 의도에 따른 도메인 자산은 자산이 가지고 있는 특성 값에 대한 부가적인 의미 때문에 Core RAS의 확장 기능을 이용하여 표현하였다[10-12]. 기존 연구에서는 자산을 계층화한 카테고리 분류방식으로 설정하였으며 아키텍처가 갖고 있는 특성을 반영하여 분류하는 방법을 적용하였다. 아키텍처 자산을 통합하는 리파지토리(repository)는 각 응용 도메인 영역에 대한 애플리케이션 구축을 위해 수집된 요구사항을 반영하기 위한 컴포넌트 자산을 검색하여 공급하기 위한 환경이다. 지원 시스템을 이용하여 자산 정보를 합성하여 이용하면 개발자는 기존 컴포넌트의 의존관계를 알아야 한다. 그리고 개발자는 이 정보를 이용하여 컴포넌트 사이의 의존관계를 분석할 수 있고 개발 절차를 진행하도록 하였다.

본 논문에서는 기존의 연구에서 복잡도의 비율이 안정화된 결과 값을 갖는 것에 비해 재사용성 비율은 어떠한 결과 값을 갖는지에 대한 측정 값 과 의미를 고찰하는데 목표로 하고 있다. 선행된 연구의 자산에 관한 복잡도를 측정한 결과로부터 얻어진 복합 자산의 평가를 기본 모델로 하여 도메인 고유의 기본자산과 응용 도메인의 복합 자산을 평가하기 위한 이원화된 평가모델을 활용하였다. 소프트웨어를 평가하기 위한 표준모델은 ISO/IEC 25010 품질 모델에 제시된 특성을 중심으로 자산이 구성하고 있는 데이터의 특성을 평가하였다[13,14]. 자산정보는 기본 자산을 확장한 복합 자산을 이용하여 설계되었을 때 각각의 자산이 갖고 있는 도메인 특성에 따라 가중치 값을 부여한 부특성을 선택적으로 평가할 수 있도록 적용하여 도메인에 따른 평가모델의 유연성을 확보하였다. 이때 개발자의 특성에 따라 나타나는 사용성과 자산 구성방법에 따른 복잡도의 변화가 어떻게 나타나는지 측정하였다. 그러나 코로나19로 인해 데이터 수집에 한계가 있어 계속적인 데이터 수집으로 측정값의 품질을 확보해야 할 것이다. 본 논문의 구성은 2절에서 자산의 구성과 측정에 대한 기반연구, 3장에서는 아키텍처 자산의 재구성 시스템에 대한 적용결과와 4장에서는 결론에 대하여 기술하였다.

2. 기반 연구

2.1 효율성 평가

자산의 효율성 평가를 위해 대상이 되는 각 자산은 설계 도메인의 특성 값과 설계자의 설계 방향에 따라 효율성 평가항목을 두 가지 영역으로 구분한다. 하나는 필수적 요소로 일반적인 효율성 요소이며 다른 하나는 선택적 요소로 설계자의 설계 의도를 선택 적용하여 평가하도록 구성하였다[7].

$$Effic = \sum_{i=1}^{n} Spec1_i + \sum_{i=1}^{n} Spec2_i \quad (1)$$

$$Spec1_i = \sum_{j=1}^{7} weight_j * Crt_j$$

$$Spec2_i = \sum_{j=1}^{3} weight_j * Crt_j$$

평가에 적용된 각각의 부특성 값은 설계 대상 시스템의 도메인 서비스에 부합되어야 하기 때문에 각각의 부특성에 대한 평가항목의 가중치를 부여하여 설계자의 의도 값을 도입한 후 이를 기반으로 효율성을 평가할 수 있도록 하였다. 효율성을 평가하기 위해 자산에 대한 각각의 평가 기준을 정하고 이때 적용하는 가중치 설정 값은 식(1)에 제시된 바와 같이 평가 대상 자산의 도메인 특성과 설계자의 유연성을 반영하기 위한 설계 의도에 따라 계수 값에 가중치를 적용하여 합산하여 평가 결과를 계산하도록 하였다.

2.2 자산평가 구성

본 연구에서 사용하는 자산 정보를 관리하기 위해 각 시스템을 평가하기 위한모델로 ISO/IEC 25010의 사용품질을 사용하여 기준을 적용하였다[13,14]. 구성된 각 자산을 평가하기 위해서는 복합자산 외에 기본적인 아키텍처 자산을 대상으로 평가 한다. 또한 기존 연구에서는 독립된 자산의 평가를 위해 각 자산에 대한 복잡성을 평가하고 이 자산들 간의 상관관계를 분석하였다. 그리고 각 자산은 아키텍처의 요구 만족을 보기 위해 기능성을 기본으로 하여 8가지의 특성을 평가하여야 한다. 복합 구성이 아닌 기본적인 자산으로 구성된 아키텍처 컴포넌트는 8가지 항목을 그대로 적용한다.

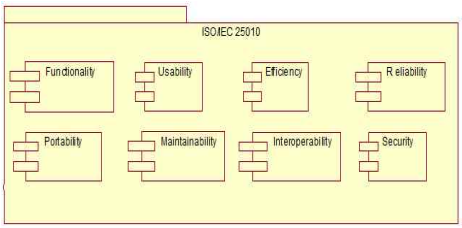

Fig. 1. ISO/IEC 25010 Configuration

자산 시스템의 품질평가를 위해 Fig 1.와 같이 ISO/IEC 25010에 언급된 필요성을 어느 정도 충족시키는지와 관련된 소프트웨어의 다양한 도메인 속성을 나타내고 있기 때문에 이를 기준 평가 방법으로 설정하여 각 자산의 평가를 위한 모델을 적용하였다. 측정하고자 하는 품질 범위에 대한 항목의 평가모델을 측정하여 기본 자산을 중심으로 복합 자산까지 전반적인 평가기반을 구축하여야 한다.

2.3 복잡도

컴포넌트는 재사용성을 기본 목적으로 하고 기능에 대한 독립성을 필요로 하는 소프트웨어 단위이므로 컴포넌트의 품질을 측정하기 위한 내부 속성 중에는 복잡도의 측정이 중요하다[14]. 복잡도는 컴퓨터를 기반으로 하는 시스템의 소프트웨어를 개발하고 유지보수하기 위한 비용에 영향을 주는 소프트웨어 고유 특성과 관련된 소프트웨어 공학의 중요한 영역이다[15]. 본 연구에서는 복잡도 측정 메트릭을 이용한다. 복잡도 측정 메트릭은 설계 단위의 컴포넌트나 실행단계의 컴포넌트에 대하여 적용할 수 있는 컴포넌트 메트릭이다. 이는 복잡도 특성을 위한 컴포넌트의 일반 복잡도에 대한 메트릭이며 클래스의 종류와 인터페이스의 합을 이용하여 복잡도를 측정하며, 이는 클래스들 사이의 메소드 복잡도를 측정하는 기준이다.

3. 자산 재구성 시스템 적용

아키텍처 자산을 목적시스템에 적용하기 위한 효율성 확보를 위해 자산을 최적화하고 사용자의 재사용성과 사용성 등에 대한 품질 확보를 위한 모델을 정형화하고 있다. 또한 이 모델 데이터들은 개발자의 숙련도와 특성에 따라 어떻게 달라지는지 고찰해보고자 한다.

3.1 자산의 정확도 변화

선행 연구에서는 자산의 최적화에 의해 재사용 효율을 높이는 경우 자산의 복잡도 변화를 측정하는데 그 목적이 있다. 선행 연구로 최적화된 자산의 정확도를 측정하였으며, 그 측정 방법과 결과는 다음과 같다.

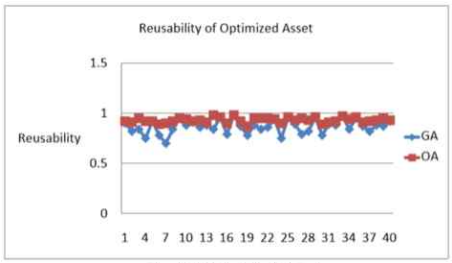

Fig. 2. Reusability Through Optimization

이때 측정을 위한 방법은 개발자의 경험에 따른 방법으로 컨텍스트 쿼리를 지원하고 개발자의 선택적 수작업을 줄여 자동으로 최적화된 자산을 검색하도록 할 수 있다. 사용하고자 하는 특성에 대한 형태로 제시된 패싯의 쿼리는 아키텍처 자산을 관리하는 시스템의 첫 번째 계층에서 계산 되어진 특성가중치와 특성-컨텍스트 관계값

을 이용하여 개발자의 쿼리와 연관된 컨텍스트 정보를 자동으로 주출하게 되어있다. 찾아진 후보자산과 쿼리 사이의 신뢰도가 계산되고, 최종적으로 신뢰도의 우선순위로 자산이 검색된다.

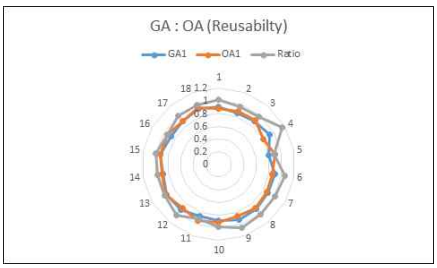

Fig. 3. Reusability of Assets

아키텍처 자산의 재사용 정확도를 측정하기 위해서 임의의 쿼리에 대한 자산 컴포넌트 재사용 정확도 결과를 측정하였다. Fig 2.에 나타난 결과와 같이 아키텍처 자산을 설계하여 저장하고 일반적으로 재사용할 때(GA)와 최적화 작업을 수행하여 재사용할 때(OA) 정확도가 증가하는 것을 알 수 있다[16]. 두 방법의 정확도의 차이는 90% 미만일 때 정확도의 증가량이 크게 증가하는 것을 알 수 있다. 이는 재사용하고자 하는 아키텍처 자산에 대한 정보를 정확히 알지 못하고 있을 때 자산의 재사용이 더 크게 증가하는 것을 의미한다. 그러나 최적화한 자산정보를 개발도상국의 개발자를 대상으로 개발자의 숙련도에 따라 적용한 결과를 나타낸 값이다. Fig 3은 개발도상국 개발자를 대상으로 재사용성을 측정한 결과 특별한 차이점을 갖지 못하는 것을 알 수 있다. 기본적인 재사용 기술의 결여와 자산들을 패싯 분류(facet classification)의 개념을 사용하여 각 자산을 하나 이상의 컨텍스트(context)로 분류하여 저장하고 유의어에 의한 확장 검색이 가능하도록 최적화되었을 때 나타나는 문제점으로 판단하고 있다. 컨텍스트 분류 방법과 유의어 선정 방법이 기술 숙련도와 개발 국가의 언어문화에 따라 영향을 받는 것을 볼 수 있다. 그러나 자산의 최적화에 의해 정확도를 높혔을 경우 복잡도의 변화를 함께 고려하여야 한다. 따라서 최적화에 의한 재사용성의 변화와 복잡도의 증감이 자산의 재사용에 대한 본질적인 목적이 훼손될 수 있다.

3.2 자산의 복잡도 변화

자산 정보는 재사용성을 기본적인 목적으로 하며 기능 독립성을 요구하는 소프트웨어 구성단위이기 때문에 자산의 품질을 측정하기 위해 반드시 필요한 자사의 내부 속성 들 중에서도 복잡도의 측정이 가장 중요하다[17]. 본 논문에서는 기존의 연구에서 제시된 순환복잡도의 측정방법을 컴포넌트에 적용하여 복잡도를 측정하는 컴포넌트의 복잡도 측정식을 자산에 적용하여 측정하였다[9]. 이 메트릭은 설계 단위의 자산이나 실행단계의 자산에 대해 적용 가능한 자산 메트릭이다. 자산의 일반 복잡도 측정을 위한 메트릭으로 적용하여 클래스, 추상화 클래스와 인터페이스의 합을 계산함으로써 자산의 복잡도를 측정하며, 클래스와 메소드의 복잡도를 측정하는 기준으로 사용할 수 있다.

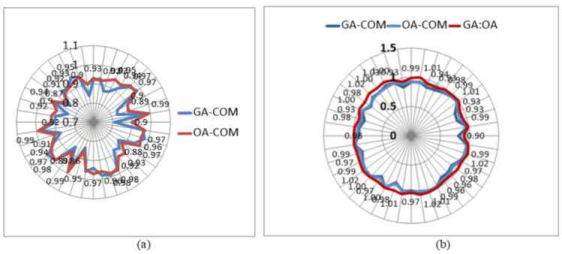

Fig. 4. Complexity Change of Assets

이상과 같이 자산의 최적화에 따라 정확도가 함께 증가한 결과에서 복잡도가 어떻게 변화하는가를 측정해 보았다(Fig 4. (a)). 측정데이터는 최적화에 사용한 자산을 대상으로 하였으며 자산을 최적화하는 과정에서 복잡도가 어떻게 변화하는가를 측정하였다(Fig 4. (b)).

$$AC_j = \frac{\sum_{i \in I_j} \mu}{|I_j|} \quad (2)$$

AC_i : set of assets
I_j : set of composite assets used by M_i assets
$\mu(i)$: number of assets using composite asset i

Fig 4에서 처럼 설계자에 의해 생성된 자산을 재사용할 때의 복잡도(GA−COM)와 최적화한 자산의 복잡도(OA−COM)의 차이는 큰 변화가 없다[15]. 그리고 자산의 최적화에 따라 복잡도 비율도 변화하지 않는 것을 알 수 있다. 식 (1)과 같이 아키텍처 자산으로 재사용되어 얻어진 복합 구성된 자산의 연관성을 측정하였으며, 이것은 자산들의 응집력에 반비례하는 것을 알 수 있었고 각 자산의 연관 값에 대한 누적된 합에 대해 비례하는 복합자산의 복잡성을 측정 하였다. Fig 5는 동일한 자산을 대상으로 개발도상국의 개발자를 대상으로 적용한 복잡도 측

정 결과이다. 설계자에 의해 생성된 자산과 최적화한 자산의 복잡도 에서는 크게 차이를 보이고 있지 않는 것으로 측정되었다. 측정 값의 상대적 복잡도에 대한 비율은 1에 수렴하고 있다는 것을 알 수 있다.

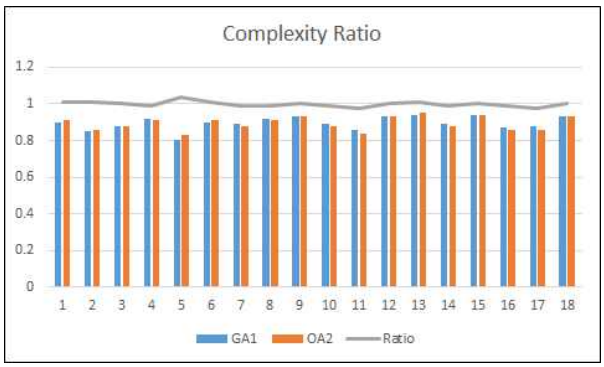

Fig. 5. Complexity Relation

따라서 자산의 복잡도 측면에는 개발도상국 개발자의 특성에 영향을 받지 않고 자산이 최적화된 것으로 판단되며 기술적 숙련도에 따른 복잡성의 변화를 살펴볼 필요가 있다.

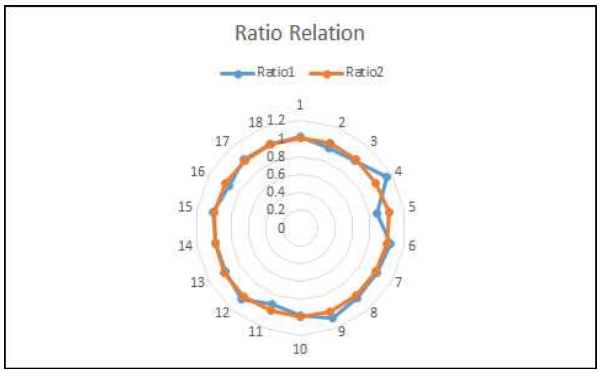

Fig. 6. Ratio Relation of Complexity

Fig 6은 본 연구에서 적용한 테스트에서 재사용 값과 복잡도 값 사이의 비율에 따른 상관관계를 나타낸 것이다. 환경적 제약에 의해 실험 데이터 값을 충분히 확보되지는 않은 결과 이지만 복잡도의 비율은 안정화된 결과 값을 나타나는 것에 비해 재사용성 비율은 불안정한 결과 값을 가질 수 있음을 알 수 있다. 이때 적용된 자산의 구성 특성과 개발 환경을 고려했을 때 패싯의 분류 특성 값이 반영되어 개발자 속성이 관여된 것으로 판단된다.

4. 결론

소프트웨어 개발은 부품화된 자산을 확보하고 이를 기반으로 개발 사이클을 단축시키고자 한다. 자산을 기반으로 하는 대부분의 시스템은 개발사의 주관적인 평가기준을 이용하여 측정되어왔다. 그리고 자산 정보를 조합하는 방식으로 하는 재사용 시스템을 평가하기 위해서 소프트웨어가 구성하고 있는 모든 품질에 대한 특성 값을 구축하고 측정하기 위해 척도 값을 다르게 적용할 수 있어야 한다. 소프트웨어 설계 단계에서 컴포넌트 자산을 기반으로 아키텍처를를 구성하는 경우에는 도메인의 속성 값이나 도메인 적용 영역에 대한 플랫폼 고유의 특성이 반영되지 않도록 독립성을 가져야 한다. 선행연구에 제시된 자산기반의 아키텍처 설계 시스템의 재사용성과 복잡도의 측정에서 나타난 결과를 이용한 복합자산을 평가하도록 하였다. 이때 자산의 평가모델은 국제 표준으로 제시된 ISO/IEC 25010의 품질에 대한 모델 특성을 이용하여 자산의 재사용 데이터에 대한 사용성을 평가하기 위한 부특성의 평가모델을 위한 기준 모델을 구축하였으며, 이번 연구에서는 개발자의 특성에 따라 기존의 측정값이 어떠한 의미를 갖는지 확인할 수 있었다. 재사용의 정확도에서 부분적으로 특징적인 값을 나타내는 것을 알게 되었으며, 그 이유는 패싯의 분류방식에서 나타나고 이것이 개발자의 특성에 따라 달라질 수 있다는 것을 확인하였다.

선행 연구에서 제안된 평가모델은 자산의 구성방식이 기본자산과 복합 자산으로 구성되어 있을 때 두 가지 자산에 대하여 모두 평가할 수 있는 모델을 정형화 하였다. 복합자산을 이용하여 아키텍처 자산을 구성할 수 있으며, 이를 이용하여 시스템이 설계되었을 때 각 자산의 도메인에 따른 특성과 설계자의 요구품질에 따라 가중치를 적용한 부특성을 선택적으로 적용한 평가를 할 수 있으므로 평가모델의 유연성을 갖도록 하였다. 평가모형을 선택적으로 적용 수 있도록 구성하였으며 이 중에서 사용성에 대한 평가모델은 도메인의 응용영역에 따라 적용하기 위한 가중치의 값을 찾기 위해 각각의 설계자산에 대한 기댓값을 학습하여 도메인에 적합한 값을 제시할 필요가 있다. 하지만 이번 연구결과로 선행 연구의 모델에서 자산정보의 제공을 위한 특성 값을 학습시킬 필요가 있다는 것을 알게 되었다.

향후 연구과제로 개발자 특성에 따라 측정값이 달라질 수 있는 패싯은 많은 양의 자산설계 정보를 학습하여 설계과정에서 얻어진 가중치를 선택 적용하는 모델에 대한 연구가 필요하다.

REFERENCES

[1] Kirill Kuprianoff, 2020). Software Optimization Problem Solver for Automated Linkage Design, European *Conference on Mechanism Science EuCoMeS 2020,* 451-458 .

[2] Altintas, N.I.Cetin, S. Dogru, A.H & Oguztuzun, H. (2011). Modeling Product Line Software Assets Using Domain-Specific Kits. *Software Engineering, IEEE Tr. 38(6),* 1376-1402.

[3] AL-Badareen AB, Selamat MH, AJabar M,Jamilah Din & Sherzod Turaev. (2011). Software Quality Models: A Comparative Study. University Putra Malaysia. *Springer Verlag Berlin Hei delberg. 119(1),* 46-55.

[4] Krahn, H., Rumpe, B., Volkel & S. MontiCore. (2010). a framework for compositional development of domain specific languages. *STTT. 12(5),* 353-372.

[5] V.B.Misic & S. Moser. (1997). Measuring Class Coupling and Cohesion : A Formal Metamodel Approach. *ASPEC'97.* 31-40.

[6] K. Phol, G. Böckle & F. van der Linden. (2010). Software Product Line Engineering: Foundations, *Principles, and Techniques: Springer.*

[7] H. Y. Choi. (2016). MetaData Structure Design of Architecture Asset in DMI. *SMB. 6(4),* 151-156.

[8] H. Y. Choi & S. H. Shim. (2015). A Study on Software Development method based on DMI. *ICSMB. 2(1).* 359-360.

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

[10] Ahlem Ben, Yousra Younes, Bendaly Hlaoui, Leila Jemni, Ben Ayed. (2014). A Meta-Model Transformation from UML Activity Diagrams to Event-B Models. *Computer Software and Applications Conference Workshops (COMPSACW) 2014 IEEE 38th International.* 740-745.

[11] Y. S. Choi & J. E. Hong, (2017). Designing Software Architecture for Reusing Open Source Software. *Journal of Convergence for Information Technology, 7(2),* 67-76.

[12] H. D. Ryu & W. J. Lee. (2012). A Study on UML based Modeling and Automatic Code Generation for Embedded Software. *Journal of Convergence for Information Technology, 2(1),* 22-40.

[13] ISO/IEC 25010. (2011). *System and Software Engineering-System and Software Quality Reauirements and Evaluation(SQuaRE)-System and Software Quality Models.*

[14] ISO/IEC 9126-1. (2011) *Software Engineering-Product Quality- Part1: Quality Model.*

[15] Simona Bernardi, José Merseguer, Dorina C. Petriu. (2012). Dependability modeling and analysis of software systems specified with UML. *ACM Computing Surveys. 45(1).*

[16] H. Y. Cho & S. H. Shim. (2017). A study on change of optimized asset complexity. *Medico-Legal Update, 18(1).* 353-358

[17] Sudha Rajesh, A. Chandrasekar. (2016). An Efficient Object Oriented Design Model: By Measuring and Prioritizing the Design Metrics of UML Class Diagram with Preeminent Quality Attributes, *IJST, 9(21), 1-9.*

최 한 용(Hanyong 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, 최적화
· E-Mail : hychoi@shinhan.ac.kr