산업융합연구 제20권 제5호 pp. 77-82, 2022
ISSN 2635-8875 / e-ISSN 2672-0124
DOI : https://doi.org/10.22678/JIC.2022.20.5.077

재구성 가능한 자산 아키텍처의 사용성 평가

최한용
신한대학교 IT융합공학부 교수

Usability Evaluation of Reconfigurable Asset Architecture

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

요 약 소프트웨어 자산을 평가하기 위해 정형화되지 않은 평가 방법으로 자산의 평가가 이루어져왔다. 본 연구에서는 기존의 소프트웨어 자산에 대한 복잡도의 측정으로부터 확보된 최적화된 자산의 사용성을 평가하고자 한다. 자산의 내부정보에 대한 논리적 복잡도를 측정하여 아키텍처의 복잡성에 대한 척도를 이용하였으며, 재사용성에 관여된 지표를 측정하여 소프트웨어 자산의 사용성과 어떠한 관계를 갖는지 평가하였다. 따라서 HVs는 두 가지 유형의 자산에 대하여 다양한 자산의 구성 방식에 따라 일정한 비율을 유지하며 적용되는 특성을 갖고 있는 것을 알 수 있다. 그러므로 자산의 사용성면에서 최적화된 자산은 가능한 다양성을 확보한 상태에서 아키텍처 설계과정에 정형성을 갖고 적용할 수 있을 것이다.

키워드 : 복합자산, 품질평가, 사용성, 아키텍처, 복잡도

Abstract Evaluating methods for software asset have been made based on subjective evaluation criteria. In this study, we try to evaluate the usability of complex assets obtained from the previous measurement of the complexity of the asset management system. The evaluation used a scale provided by measuring logical complexity to measure the complexity of the asset, and evaluated the relationship with the usability of the software asset by measuring the index related to reusability. Therefore, it can be seen that HVs maintain a constant ratio according to the composition of various assets for the two types of assets and maintain the applied consistency. Therefore, it can be determined that an asset optimized in terms of usability can be applied consistently in the architectural design process while securing as much diversity as possible.

Key Words : Composite Asset, Quality Evaluation, Usability, Architecture, Complexity

This work was supported by the Shihan University Research Fund, 2021.
*Corresponding Author : Hanyong Choi(hychoi@shinhan.ac.kr)
Received April 4, 2022 Revised May 9, 2022
Accepted May 20, 2022 Published May 28, 2022

1. 서론

소프트웨어 자산을 평가하기 위해 정형화되지 않은 평가 방법으로 자산의 평가가 이루어져왔으며, 이와 같은 소프트웨어 시스템을 평가하기 위한 방법은 주관적인 평가기준을 대상으로 이루어져왔다. 재사용 프로세스 모델에서 PO(Project Organization)는 이전의 개발과 현재의 개발로부터 얻은 모든 유형의 패키지화된 경험을 이용하여 제품을 개발하게 되며, 컴포넌트 레벨의 EF(Experience Factory)는 잠재적으로 재사용 가능한 경험들을 인지하고, PO가 사용하기 쉽도록 경험들을 패키지화한다[1]. 그리고 PO로부터 특정 요구를 받지 않고 소프트웨어 컴포넌트를 만들고자 할 때, CF(Component Factory)는 기존 시스템에서 재사용 가능한 컴포넌트들을 추출해 낸다[2-4]. 본 연구에서는 기존의 자산관리 시스템에 대한 복잡도로부터 평가된 이원 자산 아키텍처의 사용성 측면에서 여러 가지 지표를 평가하여 최적화된 자산의 특성을 분석하고자 한다[5]. 측정한 결과 값은 자산들의 환경에 따라 가중치 값을 부여하고, 이를 부특성으로 하여 유연한 평가가 가능하도록 모델의 유연성을 확보하는데 사용한다[6]. 재사용 구성 항목의 사용성(Usefulness)에 대한 주요 항목 지표를 측정하여 각 자산의 특성에 따라 나타나는 결과를 분석하였다. 따라서 분석된 결과를 기반으로 재사용성에 연관 지표를 측정하여 소프트웨어 자산의 사용성과 어떠한 관계를 갖는지 평가하여 확장된 자산의 재구성에 적용하고자 한다. 또한 도메인과 시스템 영역에서의 변화 값을 추가적으로 측정하여 사용성의 다양한 측면을 검토할 필요가 있으며, 사용성외에 비용과 품질의 측면을 함께 평가하기 위한 연구가 필요하다.

본 논문의 구성은 2장에서는 자산의 모델과 구성에 대한 관련연구, 3장에서는 프로세스 모델과 사용성, 4장에서는 평가결과 그리고 5장에서는 결론에 대하여 기술하였다.

2. 관련연구

2.1 평가구성

기존 연구에서 자산을 관리하기 위한 시스템에서는 재구성 가능한 자산을 서로 다른 도메인으로 구분하여 기본 자산과 복합 자산으로 관리하고 있다. 이와 같은 시스템은 자산의 내부 품질을 평가하기 위해서 국제 표준에 해당하는 항목을 기준으로 평가하고자 한다. 본 연구에서 자산을 관리하기 위한 시스템을 평가하기 위한 모델에 대하여 ISO/IEC 25010의 품질 기준으로 적용하였다[6,7]. 평가에 사용되는 자산은 기본적인 아키텍처 자산과 합성된 복합자산을 대상으로 평가 한다. 기존 연구에서는 자산의 독립 모듈을 평가하기 위해 모듈의 복잡성을 평가하였으며 이들의 상관관계를 분석하였다. 그리고 모든 자산의 품질을 평가하기 위해서는 효율성과 기능성 등 모든 부특성을 평가하여 종합해야 한다. 기본 아키텍처 자산은 8가지 항목을 그대로 적용한다.

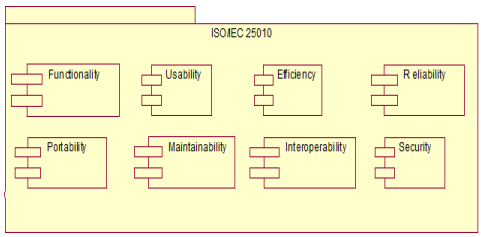

Fig. 1. ISO/IEC 25010 Configuration

본 연구에서는 모듈 구성의 자산관리 시스템에 대한 품질을 평가하기 위해 Fig. 1과 같이 국제 표준에서 나타내고 있는 필요성을 충족시키는 소프트웨어 의 특성과 특징에 대한 평가 방법을 모델로 적용하였다. 평가모델의 적용방법에 따라 모든 항목의 평가모델을 측정하여 자산의 평가모델을 선택적으로 조합할 수 있는 평가기반을 제시해야 한다. 서로 다른 도메인으로 분류된 자산에서 표준화된 아키텍처 자산은 ISO/IEC 25010의 평가 항목을 대상으로 평가하지만, 복합 자산은 도메인의 특성에서 평가하고자 하는 평가항목을 선택적으로 구성할 수 있도록 하였다[5].

2.2 재사용성 구성 항목

아키텍처 자산의 재사용성에 대한 구성항목은 사용성, 비용, 품질을 주요항목으로 갖고 있으며, 각 구성요소의 세부 요소별로 다양한 지표를 제시하고 있다. Fig. 2와 같이 재사용성에 영향을 미치는 요인들에는 크게 3종류로 분류할 수 있다[8,9].

사용성(Usefulness)은 컴포넌트에 의해서 수행되는 기능의 공통성과 다양성에 의해서 영향을 받으며, 비용(Costs)은 기존 시스템에서 컴포넌트를 추출하는 비용, 재사용 가능한 컴포넌트를 생성하는데 드는 비용, 컴포넌트를 찾고 변경하는데 드는 비용, 새로운 시스템에 통

합하는데 드는 비용을 포함한다. 그리고 품질(Quality)에는 정확성, 읽기 쉬움, 테스트 가능성, 변경의 용이함, 성능 등이 있다.

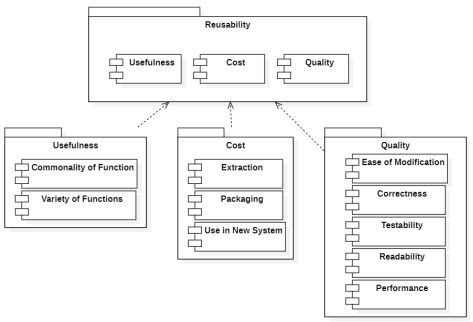

Fig. 2. Configuration of Reusability Elements

2.3 자산의 복잡도와 볼륨

컴포넌트 자산은 목적 시스템 구성할 때 재사용성을 하기 위해 기능에 대한 독립성을 필요로 하는 소프트웨어 구성단위 이며, 컴포넌트 자산의 품질을 평가하기 위해 자산 구성정보 속성 중에는 복잡도의 측정이 기본적이며 중요하다[10-12]. 복잡도는 컴퓨터에서 수행되는 소프트에어 시스템의 개발과정과 유지보수단계에서 비용에 가장 많은 부분을 차지하는 소프트웨어 고유 특성과 밀접한 소프트웨어 공학의 중요한 영역이다[12]. 본 연구에서는 복잡도 측정 메트릭을 이용한다. 복잡도를 측정하기 위해서 설계 단위의 추상화된 컴포넌트나 실행단계의 구체화된 컴포넌트를 대상으로 할 수 있다. 이 방법은 복잡도의 세부 특성을 대상으로 컴포넌트 자산의 일반적인 복잡도에 대한 측정 방법이며 클래스의 유형과 인터페이스의 합으로 복잡도를 측정한다. 이때 클래스들의 세부항목인 메소드의 복잡도는 클래스 내부 복잡도를 측정하는 기준이다.

프로그램 유닛(unit)이 자동으로 추출되고, 독립적으로 만들어진다. 또한, 프로그램 유닛은 재사용을 위해 잠재력과 관련된 관찰할 수 있는 속성에 따라 측정된다. 도메인 전문가는 컴포넌트가 향후 시스템에서 재사용 가능한지를 검사받는 동안에, 각 컴포넌트의 의미를 이해하고 기록하기 위해 후보 재사용 가능 컴포넌트들을 분석한다. 소프트웨어 자산에 대한 볼륨(Volume)의 측정 방법은 Halstead Software Science Indicator를 사용하여 측정하며, 연산자(Operator)는 프로그램의 능동(active) 요소로 산술 연산자, 판단 연산자, 할당 연산자, 함수 등 이고, 피연산자(Operand)는 프로그램의 수동적(passive) 요소로 상수, 변수 등 이다. 이때 HV(Halstead volume)은 다음 수식으로 정의된다[13].

$$HV = (N_1 + N_2)\log_2(\eta_1 + \eta_2) \quad (1)$$

$N_1 : used operators$
$N_2 : used operands$
$\eta_1 : used operator types$
$\eta_2 : used operand types$

3. 프로세스 모델과 사용성

3.1 프로세스 모델

유지보수성에서 소프트웨어의 수정 및 변경의 용이성을 측정하기 위해 모듈성, 재사용성, 분석성, 수정가능성, 시험가능성을 항목을 가지고 있다. 다수의 시스템에서 자산이 사용될 수 있으며, 그 외의 자산을 구축하기 위한 척도를 측정한 것이 재사용성이다. 본 논문의 소프트웨어 자산을 재사용하기 위한 모델구성은 Fig. 3과 같이 전통적인 생명주기 모델을 “project”와 “factory”로 분리하여 구성하였다[14,15]. 그리고 재사용 소프트웨어 자산의 동일성(identification)과 한정성(qualification)은 소프트웨어 모델과 측정값(metrics)에 기반을 둔다.

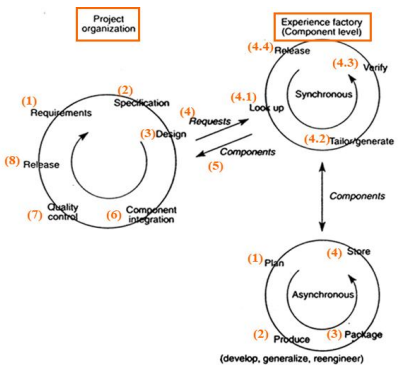

Fig. 3. Asset Composition Process Model

이전의 개발과 현재의 개발로부터 얻은 모든 유형의 패키지화된 경험을 이용하여 제품을 개발하고, 잠재적으로 재사용 가능한 경험들을 인지하고, PO(Project Organization)가 사용하기 쉽도록 경험들을 패키지화한다. 그리고 PO로부터 특정 요구를 받지 않고 소프트웨어 자산을 만들고자 할 때, CF(Component Factory)는 기존 시스템에서 재사용 가능한 자산들을 추출해낸다.

3.2 아키텍처 자산의 사용성

개발 프로세스 상에서 설계자는 저장소에 보관된 자산을 재사용할 때, 일반적인 자산의 복잡도(GA-COM:General Asset)와 도메인의 특성에 최적화된 복잡도(OA-COM:Optimized Asset)에는 의미 있을 정도의 차이는 없다. 그리고 이 두 영역에 대한 자산의 최적화에 따라 복잡도의 변화 비율도 유의미 하지 않다는 것을 알 수 있었다. 기존의 연구에서 사용한 측정 방법에서는 재사용 값과 복잡도 값에 대한 상관관계 분석을 위해 자산 간의 측정치에 대한 비율을 분석하였다. 측정을 위한 자산의 구성 방법과 도메인의 다양성을 확보하여야한다. 하지만 제약 조건에 따라 수집한 자산을 대상으로 측정한 복잡도의 비율은 모두 변화율이 적은 안정 값을 나타내고 있지만, 재사용성에 대한 비율 측정에서는 일부 자산에 대하여 불안정한 결과 값이 나타났다[5]. Fig. 4와 같이 재사용성에 영향을 미치는 요인들에는 크게 3종류로 분류할 수 있다. 본 연구에서는 이중 컴포넌트에 의해서 수행되는 기능의 공통성과 다양성에 의해서 영향을 받는 부분을 중심으로 사용성을 대상으로 하였다.

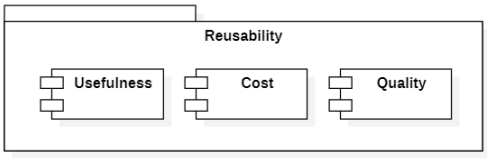

Fig. 4. Reusability Configuration

따라서 본 논문에서는 재사용성에 관여된 GA-HV, OA-HV, HV와 VoF(Variety of Function)를 측정하여 소프트웨어 자산의 사용성과 어떠한 관계를 갖는지 평가하였고 점차적으로 모든 영역을 대상으로 평가하여 검토할 필요가 있다.

4. 평가 결과

기존의 연구에서는 재사용 효율을 높이기 위해 아키텍처 자산을 최적화 하였고, 이때 자산은 그 복잡도 특성의 변화가 어떻게 달라지는지 측정하는 것이 목적이었다. 기존의 연구로 최적화된 각 자산을 대상으로 정확도의 변화를 측정한 결과는 Fig. 5와 같다.

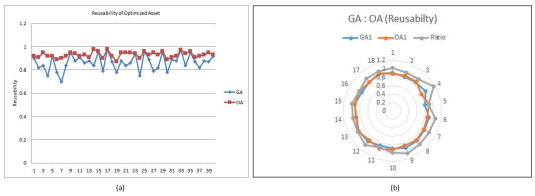

Fig. 5. Reusability of GA:OA

Fig. 5의 결과를 보면 컴포넌트를 기반으로 자산을 사용할 때, 표준화된 자산정보(GA)와 도메인에 특화된 자산(OA)을 대상으로 측정하였을 때, 재사용을 위한 자산 정보를 명확히 선택할 수 없을 때 자산의 재사용이 90%이상 상대적으로 더 크게 증가하는 것을 의미한다. 그리고 연산자(Operator)는 프로그램의 능동(active) 요소와 피연산자(Operand)는 프로그램의 수동적(passive) 요소에 의해 얻어진 HV의 변화를 측정하여 Fig. 6과 같이 일반적이 자산에 대하여 복잡도의 변화에 따라 HV값의 변화가 어떻게 영향을 받는지 확인할 수 있었다.

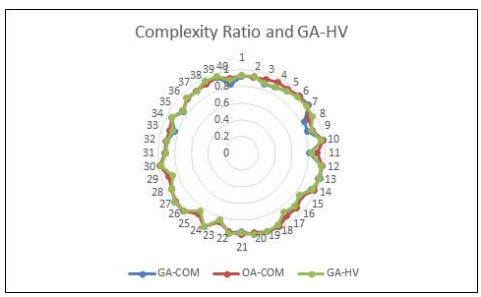

Fig. 6. Relation of Complexity and GA-HV

일반적인 자산의 복잡도에 따라 HV값은 전체적으로 균형을 이루는 것을 알 수 있으며 세부적으로는 자산을 구성한 클래스의 능동적 요소에 의해 결정에 따라 동일한 특성 값을 갖게 된다는 것을 알 수 있다. 또한 최적화된 자산(OA)에 대한 특성을 비교해 보기 위해 최적화 자산의 복잡도와 볼륨의 관계를 측정했다. 기본적으로 자산은 Operator는 프로그램의 능동(active) 요소와 Operand는 프로그램의 수동적(passive) 요소에 서 능동적 요소의 구성에 따라 복잡도의 영향을 받기 때문에 Fig. 7과 같이 최적화 자산에 대하여 복잡도의 변화에 따라 역시 동일하게 OA-HV값이 동일한 분포로 영향을 받는 것을 확인할 수 있었다.

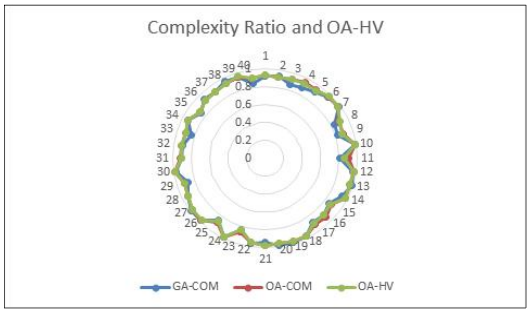

Fig. 7. Relation of Complexity and OA-HV Relation

일반적인 자산과 최적화된 자산의 복잡도에 따라 HV값은 동일하게 전체적으로 균형을 이루기 때문에 소프트웨어의 자산의 복잡도를 감소시켰을 때 자산의 사용성에서는 영향을 받지 않는 것을 알 수 있다. 또한 Fig. 8과 같이 자산이 가지고 있는 GA-HV와 OA-HV가 복잡도의 영향을 받지 않는 것을 알았지만 사용성 측면에서 중요한 요소항목인 기능의 다양성을 측정했다. 각 자산에 대한 기능의 다양성 값은 HV와 복잡도에서 나타난 특성과는 다른 양상을 보였다.

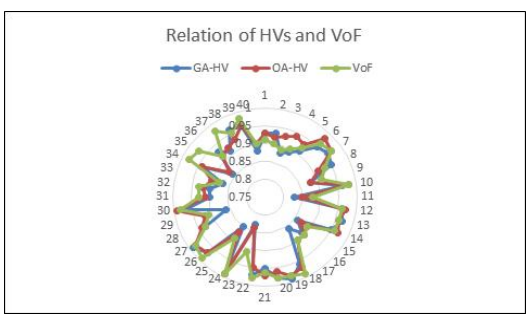

Fig. 8. Relation of HVs and VoF

대부분은 VoF가 HV와 동일한 특성 값을 갖고 있지만 HV의 값이 낮은 특성을 갖는 자상에에 또 다른 특성을 보이고 있다. 이는 HV가 복잡도와 동일한 특성값을 나타내는 것에 비해 HV에 관계 없이 자산에 대한 기능의 다양성 면에서는 그 편차가 적다는 것을 알 수 있다. 그러므로 자산의 기능에 대한 다양성은 확보하면서 재구성 가능한 자산 아키텍처를 사용할 수 있다는 것을 의미한다. 그렇다면 HV값의 변화량은 GA와 OA에 대하여 어떠한 특성을 갖게 되는지 Fig. 9에 제시하고 있다.

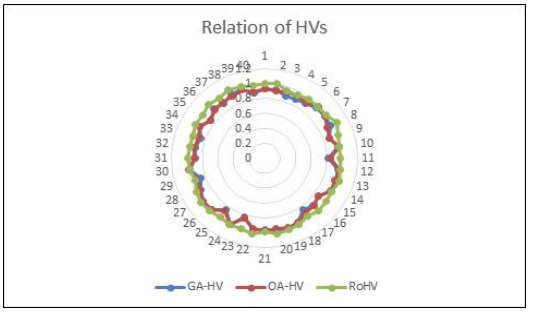

Fig. 9. Relation of HVs Ratio

GA와 OA는 최적화에 따라 복잡도에 영향을 받지 않고 사용할 수 있도록 구성되었으며, 기능의 다양성 측면에서는 HV의 영향을 덜 받도록 구성되었음을 알 수 있었다. 이때 HVs는 GA와 OA에 대하여 다양한 자산의 구성 방식에 따라 일정한 비율을 유지하며 적용되는 일관성을 유지하는 것을 알 수 있다. 그러므로 자산의 사용성면에서 OA는 기능한 다양성을 확보한 상태에서 아키텍처 설계과정에 일관적인 적용이 가능한 것으로 판단할 수 있으며 도메인과 시스템영역에서의 변화 값을 추가적으로 측정하여 사용성의 다양한 측면을 검토해 볼 수 있다.

5. 결론

기존의 연구에서 얻어진 측정값과 복잡도의 측정값 사이의 상관관계에 대한 비율을 분석하였다. 테스트를 위한 제약에 의해 일반화하기 위한 충분한 실험 데이터를 확보하여 추가적으로 측정해야 하지만 복잡도의 비율은 전반적으로 모든 자산에 대하여 유의미한 안정 값을 보였다. 하지만 상대적으로 재사용성에 대한 비율은 일부 자산에서 불안정한 결과 값을 나타낼 수 있음을 알 수 있다. 이때 재사용에 이용한 복합 자산의 구성에 대한 세부 속성과 개발 환경을 평가한 결과는 패싯의 분류 특성 값이 많은 지표 영역에 반영되어 개발자의 환경적 속성이 주로 관여된 것으로 확인 되었다. 본 논문에서는 재사용성에 관여된 GA-HV, OA-HV, HV와 VoF를 측정하여 소프트웨어 자산의 사용성과 어떠한 관계를 갖는지 평가하였다. HV값의 변화량은 GA와 OA에 대하여 어떠한 특성을 갖게 되는지 Fig. 6에 제시하고 있다. GA와 OA는 최적화에 따라 복잡도에 영향을 받지 않고 사용할 수 있도록 구성되었다. 이때 HVs는 GA와 OA에 대하여 다양한 자산의 구성 방식에

따라 일정한 비율을 유지하며 적용되는 일관성을 유지하는 것을 알 수 있다. 그러므로 자산의 사용성면에서 OA는 가능한 다양성을 확보한 상태에서 아키텍처 설계과정에 일관적인 적용이 가능한 것으로 판단할 수 있다. 향후 연구과제로 도메인과 시스템 영역에서의 변화값을 추가적으로 측정하여 사용성의 다양한 측면을 검토할 필요가 있으며, 사용성외에 비용과 품질의 측면을 함께 평가하기 위한 추가 연구가 필요하다.

REFERENCES

[1] V. Basili, G. Caldiera, F. McGarry, R. Pajerski, G. Page & S. Waligora. (1992). The software engineering laboratory: an operational software experience factory. *In Proceedings of the 14th international conference on Software engineering,* 370-381.

[2] N. I. Altintas, S. Cetin, A. H. Dogru & H. Oguztuzun. (2011). Modeling Product Line Software Assets Using Domain-Specific Kits. *IEEE Tr, 38(6),* 1376-1402. DOI: 10.1109/TSE.2011.109

[3] 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, 12(5),* 53-372.

[4] K. Phol, G. Böckle & F. van der Linden. (2010). *Software Product Line Engineering: Foundations, Principles, and Techniques,* (Vol. 1). Springer.

[5] 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, (S.I 08),* 143-149.

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

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

[8] N. Padhy, S. Satapathy & R. Singh. (2017). Utility of an Object Oriented Reusability Metrics and Estimation Complexity. *Indian J. Sci. Technol, 10(3),* 1-9. DOI: 10.17485/ijst/2017/v10i3/107289

[9] S. Bernardi, J. Merseguer & D. C. Petriu. (2012). Dependability modeling and analysis of software systems specified with UML. *ACM Computing Surveys, 45(1),* 1-48. 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. *In 2014 IEEE 38th International Computer Software and Applications Conference Workshops,* 740-745.

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

[12] 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. DOI: 10.22156/CS4SMB.2017.7.2.067

[13] T. Hariprasad, G. Vidhyagaran, K. Seenu & C. Thirumalai. (2017). *Software complexity analysis using halstead metrics. In 2017 International Conference on Trends in Electronics and Informatics (ICEI)* (pp. 1109-1113). IEEE.

[14] Y. Liu, L. Zhang, W. Zhang & X. Hu. (2016). An overview of simulation-oriented model reuse. *Theory, methodology, tools and applications for modeling and simulation of complex systems,* 48-56.

[15] S. 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, *Indian Journal of Science and Technology,* 9(21).

최 한 용(Hanyong Choi) [정회원]

· 1993년 2월 : 경희대학교 전자계산공학과(공학사)
· 1998년 2월 : 경희대학교 전자계산공학과(공학석사)
· 2002년 8월 : 경희대학교 전자계산공학과(공학박사)
· 2004년 3월~현재 : 신한대학교 컴퓨터공학전공 교수
· 관심분야 : 아키텍처, 플랫폼 디자인, 품질관리, 인공지능
· E-Mail : hychoi@shinhan.ac.kr