KIPS Tr. Software and Data Eng.
Vol.5, No.10 pp.465~470 pISSN: 2287-5905

http://dx.doi.org/10.3745/KTSDE.2016.5.10.465

Study for Architecture Environment Consideration to Software Interaction

Eun-Ser Lee†

ABSTRACT

Project success is depend on requirement analysis and design for in the software engineering. Architecture error have a effect in the whole system. As a result, the software quality will deteriorate. Therefore, we are need to check that software interaction and modify for a stable architecture and environment in the design phase. In this paper, we are suggest that factors of the architecture environment consideration.

Keywords : Design of Architecture, Quality of Architecture, Environment of Architecture

소프트웨어 연동을 위한 아키텍처 환경 고려사항에 관한 연구

이 은 서†

요 약

소프트웨어 공학에서 요구사항 분석은 전체 시스템의 성공률을 좌우한다. 아키텍처에서 발생되는 오류는 전체 시스템에 영향을 주게 되고, 그 결과 소프트웨어 만족도가 낮아진다. 따라서 설계 단계에서 안정적인 아키텍처와 환경을 위하여 소프트웨어 간의 상호작용과 수정할 수 있는 확인이 필요하게 된다. 본 논문에서는 아키텍처 환경 고려 사항의 요소들을 제안하고자 한다.

키워드 : 아키텍처 설계, 아키텍처 품질, 아키텍처 환경

1. 서 론

소프트웨어 개발 과정에서 요구사항 분석 후, 설계를 수행한다. 설계는 요구사항 분석 내용을 기반으로 진행하게 된다. 따라서 요구사항과의 연속성과 일관성이 유지되어야 한다. 이와 같은 사항은 설계의 품질에 많은 영향을 끼치게 된다.

설계에서는 여러 가지 방법론을 사용하여 고객의 요구사항을 시스템화 하기 위하여 부단히 노력하게 된다. 그 과정에서 아키텍처를 설계하게 된다. 아키텍처는 여러 소프트웨어들이 잘 연동될 수 있는 틀을 제공한다. 따라서 아키텍처의 품질을 측정하고 구현하는 것은 소프트웨어 개발 과정에서 중요한 요인이 되고 있다.

프로그램 또는 전산 시스템의 소프트웨어 아키텍처는 소프트웨어 컴포넌트, 이들 컴포넌트의 외부적으로 보이는 속성 및 이들 사이의 관계로 이루어진 시스템의 구조이다. 아키텍처는 운용 소프트웨어가 아니다. 아키텍처는 다음사항을 표현한다[1, 3, 4, 10, 11].

- 기술 한 요구사항을 충족시키는 데 있어서 설계의 효과성을 분석한다.
- 설계변경이 여전히 상대적으로 용이한 단계에서 아키텍처 대안을 고려한다.
- 소프트웨어 구축에 관련된 위험성을 감소시킨다[7-9].

아키텍처는 설명과 같이 소프트웨어 구축에 중요한 요인이 된다. 본 논문에서는 아키텍처에 의한 소프트웨어 연동시 환경적인 고려사항을 제안한다. 제안된 사항을 아키텍처 설계 전에 수행하여 구축하려는 소프트웨어를 어떤 형태로 지원할 것인가와 필요한 기술들이 잘 지원되는가를 확인하여 아키텍처 품질을 향상시킬 수 있게 된다.

2. 기반 연구

2.1 소프트웨어 아키텍처 정의

소프트웨어 아키텍처는 소프트웨어 시스템을 구현하기 이

※ 이 논문은 2016학년도 안동대학교 연구비에 의하여 연구되었음.
† 종신회원 : 안동대학교 컴퓨터공학과 부교수
Manuscript Received : September 8, 2016
Accepted : September 22, 2016
* Corresponding Author : Lee Eun-Ser(eslee@anu.ac.kr)

전에 소프트웨어 시스템에 관한 많은 중요한 결정을 내릴 수 있게 하고, 아키텍처 수준에서 소프트웨어 시스템에 대한 분석을 가능하게 함으로써 계획된 소프트웨어 개발이 가능하도록 한다. 소프트웨어 아키텍처가 아키텍처 설계에 필요한 형태로 요구사항이 철저히 분석될 수 있도록 가이드 하는 한편, 아키텍처설계 단계에서 내려진 소프트웨어 구조를 포함하나 제반 아키텍처적인 결정들이 상세 설계, 구현, 통합 및 시험에까지 집행될 수 있도록 가이드 하는 구체적인 산출물임을 보여준다. 소프트웨어 시스템은 개발 이후에도 많은 시간과 비용을 들여 진화하게 되는데, 아키텍처는 소프트웨어 진화에도 영향을 미친다. 따라서 소프트웨어 아키텍처에서는 품질과 연관되어 분석, 설계 과정이 진행되어야 한다[14].

아키텍처의 품질 속성을 고려하여 설계가 진행되어야 한다.

설계와 연관하여 패턴에서는 자주 발생하는 '문제'에 대한 준비된 해결방법이므로, 대개의 경우 특정한 품질을 확보하는 문제에 대한 해결방법을 제시하지는 않는다. 패턴을 통하여 품질 개선 효과가 얻어질 수도 있으나 그것이 만족스럽지 않을 수도 있다. 일반적으로 품질속성은 성능, 상호운영성, 변경용이성, 가용성, 보안성, 시험용이성 및 사용용이성의 일곱 개의 속성으로 제시되고 있다[15, 16].

특정 품질속성에 대한 설계전략의 도출 절차는 다음과 같다.

- 관련 품질속성을 위한 분석모델로 시작한다.
- 모델의 파라미터를 식별한다.
- 모델의 마라미터를 조정하기 위한 아키텍처의 기법을 식별한다.

2.2 결함 제거 효율(DRE : Defect Removal efficiency)

프로젝트 수준과 프로세스 수준 모두에 유익한 품질 계량은 결함 제거 효율이다. 근본적으로 결함 제거 효율은 전반적인 프로세스에 걸친 품질 보증과 품질제어 행위에서 걸러내는 능력에 대한 측정이다. 프로젝트들은 총체적으로 고려할 때, 결함 제거 효율은 다음과 같이 정의된다[2, 5, 6].

$$DRE = E / E + D$$

E : 소프트웨어가 최종 사용자에게 배포되기 이전에 발견된 오류의 수
D : 배포된 후에 발견된 결함의 수

이상적인 DRE의 값은 1이다. 소프트웨어에서 결함이 발견되지 않았다는 것을 의미하며 현실적으로 D는 0보다 크고 E가 주어진 D의 값에 따라 증가하므로 DRE의 값은 역시 1에 가까워진다. E의 값이 증가하면 D의 최종 값은 감소한다. 품질 제어 및 품질 보증 활동의 걸러내는 능력에 대한 지표로 계량(metrics)이 사용될 경우, DRE는 소프트웨어 프로젝트 팀이 소프트웨어의 배포 이전에 가능한 한 많은 오류를 걸러낼 수 있게 하는 기법을 제정하도록 유도한다.

DRE는 또한 프로젝트의 수행 중에 오류들이 다음 단계, 또는 다음 소프트웨어 공학 활동에 넘어가기 전에 찾아내는 능력을 평가하는 데에도 사용될 수 있다[12, 13].

$$DRE = Ei / Ei + E_{i+1}$$

E_i : 소프트웨어 공학 활동에서 i에서 발견된 오류의 수
E_{i+1} : 소프트웨어 공학 활동에서 i+1에서 발견된 오류의 수

i+1 단계에서 발견된 오류들은 소프트웨어 공학 활동 i에서 발견되지 않은 오류라고 추적할 수 있다.

3. 본 론

아키텍처는 문제를 해결하고 완성도 있는 소프트웨어 개발을 위하여 다양한 형태로 존재한다. 이와 같은 과정에서 중요한 요인으로 나타나는 것이 아키텍처 품질의 문제이다.

아키텍처에서 실행되는 소프트웨어는 다른 소프트웨어와 연동이 되게 된다. 연동되는 소프트웨어는 유형은 다음과 같다.

- 개발하는 과제의 결과물로서 만들어진 소프트웨어
- 레거시 소프트웨어

개발된 소프트웨어 중에서도 오류를 수정한 소프트웨어의 경우, 레거시 소프트웨어 또는 시스템과 상호 연동에서 문제가 발생된다. 이와 같은 경우에는 오류를 수정한 소프트웨어의 내부 알고리즘과 인터페이스의 변동을 분석하여 레거시 소프트웨어와 시스템에 영향을 없는지를 파악해야 한다. 많은 소프트웨어 연동에서 오류가 발생되는 부분이 레거시 시스템과 소프트웨어의 경우 이미 사용하여 검증이 되었고 새로운 기능만 구현하여 연동을 하므로 문제가 없다고 판단을 하게 된다. 이와 같은 상황에서 새로운 기능이 레거시 소프트웨어의 기능과 연동이 되는 형태를 사용자가 분석 및 설계에서 인지하지 못하는 경우에, 문제가 발생되고 발생된 문제의 원인을 발견할 수 없게 되어서 전체 시스템에 영향을 주게 된다.

소프트웨어의 상호연동을 고려하여 시스템에 적용하기 위해서는 아키텍처 설계 시에 고려되어야 할 사항이 있다.

고려사항은 연동되는 환경적인 관점으로 플랫폼, 데이터베이스, 미들웨어로 제시하였다.

3.1 소프트웨어 연동을 위한 환경 관점의 고려사항

소프트웨어의 상호연동을 위한 환경 관점의 고려사항을 플랫폼, 데이터베이스, 미들웨어로 제시하였다.

- 플랫폼 요소

플랫폼은 소프트웨어의 연동을 위하여 확인해야 할 사항이다. 플랫폼 간에는 서로 호환이 되는 경우도 있다. 그리고 같은 플랫폼이라도 버전에 따라서 연동이 불가한 경우도 빈번히 발생하게 된다.

플랫폼에서는 기능의 확장성, 네트워크를 통한 연동, 각 요소에서의 정보 공유, 공유할 데이터에 대한 무결성 유지 측면에서 구분하여 제시하였다.

Table 1. Items of Confirmation View Point of Platform

Elements	Contents
Extensions of function	Check the extensions of function
Probability of network connection	Check the probability of network connection
Sharing information	Probability of sharing information
Data integrity	Check the data integrity

- 기능 확장성

플랫폼의 고유 기능 중에서 확장된 기능을 수용할 수 있는지를 확인하는 것이다. 기능 확장성의 확인을 통하여 아키텍처 설계시에 확장성을 고려하여 소프트웨어를 구성할 수 있게 된다. 만약, 플랫폼에서 기능 확장성을 제공하지 않는다면 아키텍처에서 새롭게 만든 기능이나 다른 소프트웨어의 연동을 위한 기능들을 제공할 수 없게 된다. 따라서 아키텍처 품질에서 확인해야할 중요한 요인이 된다.

- 네트워크 연결 가능성

플랫폼의 네트워크 연결 가능성은 아키텍처에서 구동되는 소프트웨어가 네트워크로 상호연동이 가능하게 된다. 이와 같은 내용을 확인하는 것은 아키텍처 설계 시, 소프트웨어 구성을 네트워크가 가능하게 할 수 있다. 또한 아키텍처의 수정 및 연동을 네트워크로 해결할 수 있게 된다.

- 정보 공유

정보 공유는 플랫폼에 연동되는 요소간의 정보 공유가 가능한지를 확인하는 것이다. 플랫폼에서 정보의 흐름을 허용하지 않는다면 아키텍처 구성에서 정보 공유를 기반으로 소프트웨어가 연동되는 형태를 구성할 필요가 없다. 따라서 아키텍처에서 소프트웨어 상호연동 형태를 결정할 수 있게 된다.

- 데이터 무결성

데이터 무결성은 다른 아키텍처에 가공되거나 수정되지 않는 제공할 수 있는지에 관하여 확인하는 것이다. 데이터 무결성은 플랫폼과 데이터베이스, 미들웨어에서도 고려되어야 할 사항이다.

● 데이터베이스 요소

데이터베이스는 소프트웨어의 연동 시에 사용될 데이터를 관하여 중요한 요소가 된다. 소프트웨어 연동에서 데이터를 기반으로 구동되기 때문이다. 따라서 데이터베이스에 연관된 요소를 아키텍처 설계 시에 확인하여 품질을 높이고자 한다. 확인해야 할 사항으로는 소프트웨어 연동 시, 직접적인 연결을 결정하는 사용자 인터페이스 연동성, 사용될 데이터를 활용할 수 있는 데이터 조작 가능성, 상호연동 되는 소프트웨어의 관계 정의를 위한 상호연동 관계정의, 소프트웨어의 연동에서 데이터베이스 구조의 변경이 요구될 때 수용할 수 있는지에 관한 데이터베이스 구조 변경 가능성, 데이터의 무결성을 제공하여 다른 소프트웨어에서도 안전한 데이터를 제공받을 수 있는지에 관한 데이터 무결성으로 제시하였다.

Table 2. Items of Confirmation View Point of Database

Elements	Contents
Interoperability of user interface	Check the interoperability of user interface
Probability of data manipulate	Check the probability of data manipulate
Definition of interworking relationship	Check the definition of interworking relationship
Probability of change database structure	Check the probability of change database structure
Data integrity	Check the data integrity

- 사용자 인터페이스 연동성

사용자 인터페이스 연동성은 소프트웨어 연동 시, 직접적인 연결을 결정하는 요소인 사용자 인터페이스 간의 문제점을 확인하는 것이다. 사용자 인터페이스는 소프트웨어의 연동이 잘 되는지 확인할 수 있는 중요한 요인이다.

- 데이터 조작 가능성

데이터 조작 가능성은 소프트웨어 연동 시, 사용될 데이터의 조작이 필요한 경우에 확인을 하는 것이다. 데이터는 활용에 따라서 정렬과 탐색 등이 필요한 경우가 있다. 이러한 경우에 데이터 조작이 가능해야 활용하게 된다.

- 상호연동 관계정의 가능성

상호연동 관계정의 가능성은 상호연동 되는 소프트웨어 간의 관계 정의하여 관계의 수, 연결 관계를 위한 조건을 정의할 수 있어야 한다. 해당 기능이 주어지는 경우 연결 오류를 줄일 수 있다.

- 데이터베이스 구조 변경 가능성

데이터베이스 구조 변경 가능성은 소프트웨어의 연동에서 데이터베이스 구조의 변경이 요구될 때 수용할 수 있는지에 관한 확인이다.

- 데이터 무결성

데이터의 무결성을 제공하여 다른 소프트웨어에서도 안전한 데이터를 제공받을 수 있는지에 관한 확인이다.

● 미들웨어 요소

미들웨어 요소는 소프트웨어 연동 시, 분산 컴퓨팅 환경에서 연동이 생성되는 경우이다. 최근 네트워크의 발전에 따라서 소프트웨어의 연동이 외부 네트워크를 활용하여 수행되는 경우가 대부분의 요소이다. 따라서 분산 컴퓨팅 환경에서 다양한 하드웨어, 네트워크, 프로토콜, 응용프로그램 등 이 기종간의 연동을 확인할 필요가 있다.

미들웨어 요소에서는 소프트웨어 연동을 확인하기 위하여 소프트웨어가 구동되는 하드웨어 간의 연동이 가능한가의 하드웨어 호환성, 네트워크를 통해서 소프트웨어가 연동이 되므로 네트워크 프로토콜 호환성, 이 기종 간의 데이터 교환을 통한 연동을 확인하기 위하여 이 기종 데이터 호환성, 미들웨어를 통한 데이터 교환 시. 무결성을 제공할 수 있는지의 데이터 무결성을 제시하였다.

Table 3. Items of Confirmation View Point of Middleware

Elements	Contents
Compatibility of hardware	Check the compatibility of hardware
Compatibility of network protocol	Check the compatibility of network protocol
Compatibility of heterogeneous data	Check the compatibility of heterogeneous data
Data integrity	Check the data integrity

- 하드웨어 호환성

하드웨어 호환성은 미들웨어를 통하여 소프트웨어가 연동될 때, 하드웨어에 의하여 수행되게 된다. 따라서 하드웨어의 연동(네트워크 카드, 그래픽 카드 등)에서 문제가 없는지 확인하는 것이다. 기본적으로 하드웨어에서 호환성이 제공이 되어야 소프트웨어 간에 통신을 수행할 수 있게 된다.

- 네트워크 프로토콜 호환성

네트워크를 통하여 소프트웨어가 연동되는 경우, 네트워크 간의 연결이 기본으로 제공되어야 한다. 따라서 네트워크 간의 호환이 되는지를 확인하는 것이 네트워크 프로토콜 호환성이다.

- 이 기종 데이터 호환성

소프트웨어 연동을 위하여 데이터가 필요한데 데이터는 기종에 따라서 데이터의 형태가 결정된다. 따라서 다른 종류의 데이터가 호환될 수 있도록 미들웨어에서 이 기종간의 데이터 교환을 제공하는지를 확인하는 것이다.

- 데이터 무결성

미들웨어에서 제공되는 데이터가 변형되거나 가공되지 않은 내용을 제공할 수 있는지를 확인하는 것이다. 데이터 무결성은 모든 시스템에서 기본으로 제공이 되어야 소프트웨어 연동에서 생성되는 데이터가 신뢰성이 있게 된다.

4. 적용 및 사례

4.1 소프트웨어 연동을 위한 환경 관점의 고려사항 적용 사례

3장에서 제시한 이론을 기반으로 사례에 적용하였다.

적용한 사례에서 아키텍처 설계 시, 3장의 체크 항목을 사전에 확인하여 실제 구현되는 구현물의 오류를 줄이고자 하였다. 적용한 체크 항목의 실효성은 결함제거효율성으로 확인하였다.

적용한 사례는 분산된 사용자의 의사 결정을 하기 위하여 네트워크로 의사결정을 할 수 있는 도구를 설계하였다. 요구사항은 다음과 같다.

- 저장하기 - 투표결과를 저장한다.
- 불러오기 - 저장한 데이터를 불러온다.
- item설명 - item에 대해 설명한다.
- 기능 설명 - 각 기능에 대해 설명한다.
- 메시지기능 - 질의응답을 가능하게 한다.
- 기능우선순위 - 기능들의 우선순위를 투표한다.
- 만족도투표 - 프로그램의 만족도에 대해 투표한다.

요구사항을 기반으로 하여 유스케이스를 작성하였다. 유스케이스는 다음과 같다.

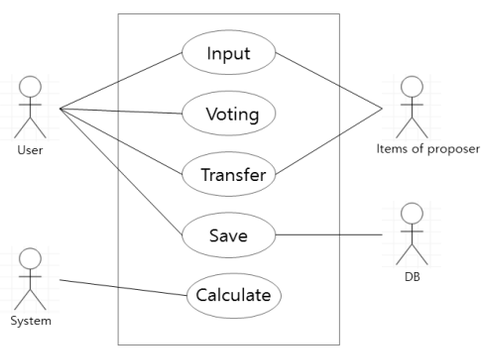

Fig. 1. Use Case Diagram

유스케이스를 작성한 후에 아키텍처를 설계한다. 아키텍처 설계를 수행하기 전에 3장에서 제시한 사항들을 확인하게 된다.

Table 4. Incase of Platform in Case

Elements	Contents
Extensions of function	Support of extensions of function in the all of platform
Probability of network connection	Can be connected network
Sharing information	Can be sharing information
Data integrity	Support of data integrity

Table 5. Incase of Database

Elements	Contents
Interoperability of user interface	Interoperable user interface
Probability of data manipulate	Can be data manipulate
Definition of interworking relationship	Can be definition of interworking relationship
Probability of change database structure	Can be change database structure
Data integrity	Support of data integrity

Table 6. Incase of Middleware

Elements	Contents
Compatibility of hardware	Support the compatibility of hardware
Compatibility of network protocol	Support the compatibility of network protocol
Compatibility of heterogeneous data	Support the compatibility of heterogeneous data
Data integrity	Support of data integrity

Table 4, Table 5, Table 6에서는 3장의 이론을 근거로 하여 사례에 적용하여 조사하였다. 이를 기반으로 시퀀스 다이어그램과 클래스 다이어그램을 작성하였다.

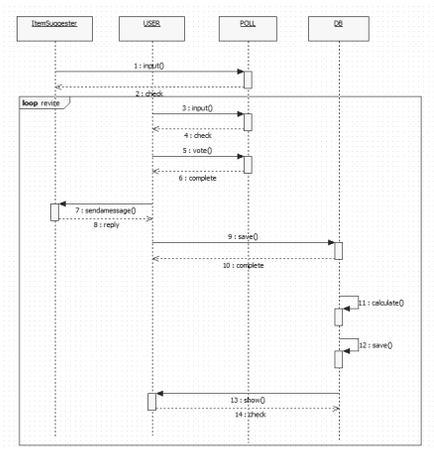

Fig. 2. Sequence Diagram

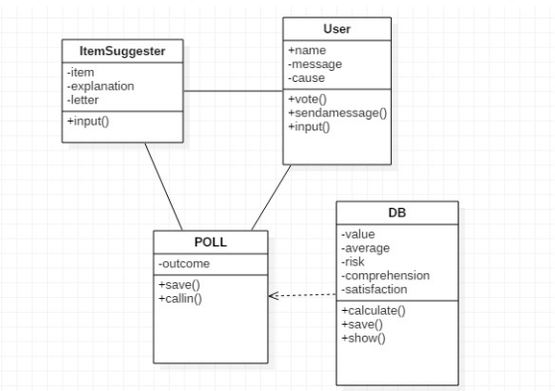

Fig. 3. Class Diagram

4.2절에서는 3장의 이론을 적용하기 이전의 결함제거효율과 이론을 적용한 결함제거효율을 비교하여 논문의 이론을 검증한다.

4.2 검증

4.1절에서 설계된 내용에 대하여 효율성을 검증하고자 한다. 검증 방법은 2.2절에서 소개한 결함제거효율(DRE) 방법을 활용하였다.

검증 방법은 설계된 도구를 적용하기 전의 DRE와 도구를 적용한 후의 DRE 결과를 비교하고자 한다.

3장의 이론을 적용하기 전의 요구사항 분석 단계와 설계 단계의 결함 개수를 산정하였다.

Table 7. Number of Defect (Requirements, Design)

	Ei	Ei+1
Requirement phase	13	12
Design phase	17	11

결함 개수를 기반으로 DRE를 산출하면 다음과 같다.
요구사항 분석 단계 : 13/13+12 = 0.52
설계 단계 : 17/17+11 = 0.607

다음은 3장의 이론을 적용한 후의 결함 개수를 산정하였다.

Table 8. Number of Defect (Requirements, Design)

	Ei	Ei+1
Requirement phase	13	7
Design phase	17	6

결함 개수를 기반으로 DRE를 산출하면 다음과 같다.
요구사항 분석 단계 : 13/13+7 = 0.65
설계 단계 : 17/17+6 = 0.739

결과를 분석하면 이론을 적용한 후의 결과가 적용 전보다 결함제거효율이 향상되었다.

5. 결 론

본 논문에서는 소프트웨어 연동을 위한 환경 관점의 고려사항을 제시하고 사례에 적용하여 효율성을 검증하였다. 소프트웨어 연동을 위한 환경 관점의 고려사항은 플랫폼, 데이터베이스, 미들웨어 관점으로 제안하였다. 따라서 제안 내용을 활용하여 아키텍처 품질을 향상시키고자 한다. 향후 연구로는 소프트웨어 연동을 위하여 내부적인 고려사항에 대하여 연구할 필요가 있다.

References

[1] Yoon chung, “Successful Software development methodogy,” Life power press, 1999.08.
[2] Huh won sil, “System analysis and design,” Hanbit media, 2006.05.
[3] Software process improvement forum, KASPA SPI-7, 2002.
[4] J. A. McCall, P. K. Richards, and G. F. Walters, “Factors in Software Quality,” vol. 1, 2, and 3, AD/A-049-014/015/055, Springfield, VA: Nat'l Tech. Information Service, 1977.

[5] Wohlin Runeson, “Defect content estimations from review data,” *Proceedings international conference on software engineering ICSE*, pp.400-409, 1998.
[6] Gaffney John, “Some models for software defect analysis,” in *Lockheed Martin Software Engineering Workshop*, 1996.
[7] L. hatton, “Is modularization always good idea,” *Information and Software Technology*, Vol.38, pp.719-721. 1996.
[8] B. compton and C. withrow, “Prediction and control of ada software defects,” *J. Systems and Software*, Vol.12, pp.199-207, 1990.
[9] Roger S. Pressman, “Software engineering,” 8th edi., Mcgraw-hill international edition, 1997.
[10] Choi Eun Man, “Software Engineering,” Jungik publishing co, 2011.
[11] Charles conrick IV and Scott hanson, “Vertical option spreads: a study of the 1.8 standard deviation inflection point,” Hoboken, N.J. : Wiley, 2013.
[12] Lan Sommerville, “Software engineering,” Addison Wiley, 2008.
[13] Shari Lawrence Pfleeger. Joanne M. Atlee, “Software engineering theorey and practice,” Pearson, 2013.
[14] Kang Sungwon, “Invitation to software architecture,” Hongrung publishing, 2015.
[15] D. Garlan and D. Perry, “Introduction to the special issue on software architecture,” *IEEE Transactions on Software Engineeriing*, 1995.
[16] L. Bass, P. Clements, and R. Kazman, “Software Architecture in practice,” 3rd ed., Addision-wesley, 2013.

이 은 서

e-mail : eslee@anu.ac.kr
2001년～현 재 ISO/IEC 15504 국제 선임 심사원
2004년 중앙대학교 컴퓨터공학과(박사)
2004년～현 재 임베디드 산업협회 전문위원
2004년～현 재 한국정보통신기술협회 위원
2012년～현 재 안동대학교 컴퓨터공학과 부교수
관심분야 : CBD, Formal method, Quality model, SPI(Defect Analysis)