KIPS Tr. Software and Data Eng.
Vol.5, No.10 pp.471~476 pISSN: 2287-5905

http://dx.doi.org/10.3745/KTSDE.2016.5.10.471

Study for Confirmation of Configuration Component of Architecture Interaction

Eun-Ser Lee†

ABSTRACT

Software architecture is depend on software quality for in the design phase. Architecture interoperability have a effect in the software quality. As a result, the software quality will deteriorate. Therefore, we are need to check that configuration component for a flexible architecture and quality in the architecture. In this paper, we are suggest that configuration component of the architecture interaction.

Keywords : Software Architecture, Design of Architecture, Software Quality

소프트웨어 연동을 위한 아키텍처간의 구성요소 확인에 관한 연구

이 은 서†

요 약

설계 단계에서 소프트웨어 아키텍처는 소프트웨어 품질을 좌우한다. 아키텍처 연동은 소프트웨어 품질에 영향을 주게 된다. 그 결과 소프트웨어 만족도가 낮아진다. 따라서 아키텍처에서 유연한 아키텍처와 품질을 위하여 구성요소 확인이 필요하게 된다. 본 논문에서는 아키텍처 연동의 구성 요소들을 제안하고자 한다.

키워드 : 소프트웨어 아키텍처, 아키텍처 설계, 소프트웨어 품질

1. 서 론

아키텍처 설계는 기능적 시스템 요구사항과 비기능적 시스템 요구사항을 만족시키도록 시스템 구성을 설정하기 위해 노력하는 프로세스이다. 아키텍처 설계가 프로세스 내부의 활동들이 개발 중인 시스템의 유형, 시스템 아키텍트의 배경과 경험, 시스템의 특정 요구사항에 따라 근본적으로 다르기 때문이다[1, 4, 7, 8].

프로젝트 시작 초기에는 복잡도가 높지 않을 것으로 예상했는데, 시간이 지날수록 복잡도가 기하급수적으로 커지는 경우가 있다. 대규모 프로젝트는 시작하기 전에 철저하게 준비하고 약속도 잘 지켜야 한다. 그리고 사용자의 요구를 만족시키려면 적시성, 유연성, 통합 등의 특성을 갖추어야 한다[14-16].

소프트웨어 아키텍처는 외부에서 인식할 수 있는 특성이 담긴 소프트웨어의 골격이 되는 기본구조로, 시스템 전체에 대한 큰 밑거름이다. 소프트웨어 아키텍처의 특성은 다음과 같다[9-11, 17].

- 개발할 소프트웨어에 대한 전체적인 구조를 다룬다.
- 소프트웨어를 이루고 있는 여러 구성 요소를 다룬다.
- 구성 요소들이 인터페이스를 통해서 어떻게 상호작용하는지를 정의해야 한다.
- 세부 내용보다는 중요한 부분만을 다룬다.
- 시스템 설계와 개발 시 적용되는 원칙과 지침이 있어야 한다.
- 의사소통 도구로 활용할 수 있어야 한다.
- 구현에 대한 제약 사항을 정의해야 한다.
- 품질 속성을 결정해야 한다.
- 재사용할 수 있게 설계해야 한다.

이와 같은 특성은 소프트웨어 아키텍처와 연동이 가능한 소프트웨어간에도 적용이 된다.

본 논문에서는 아키텍처와 소프트웨어간의 연동 시 확인해야 할 구성요소에 대하여 제시를 한다.

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

2. 기반 연구

2.1 아키텍처 모델링

아키텍처를 모델링하는데 있어서 아키텍처의 어떤 속성들은 외부에서 보여 질 수 있도록 표현되는 반면, 다른 속성들을 감추려 한다. 이런 방법으로 시스템의 다른 국면들에 의해 방해 받지 않고 필요한 속성에 대해 잘 알 수 있게 된다. 가장 주요한 것은 이런 모델들의 모임들 덕분에 개발자는 제안된 아키텍처가 명시된 요구사항에 적합한지를 추론할 수 있다는 것이다[3]. 아키텍처 모델이 사용될 수 있은 여섯 가지 방법은 다음과 같다[3].

- 시스템을 이해하기 위해 이용된다.
- 시스템이 얼마나 기존에 만들어진 다른 시스템들의 요소들을 재사용할 것인지 또 개발되는 시스템이 미래에 얼마나 재사용 가능한지를 판단하는데 이용한다.
- 시스템의 골격을 지지하는 부분을 포함한 시스템 구축을 위한 청사진을 제공한다.
- 성능, 가격, 프로토타이핑에 관한 고려 사항 등을 포함한 시스템의 진화 방법에 대해 추론하는데 이용된다.
- 종속성을 분석하여 가장 적당한 설계와 구현 그리고 테스팅 기법을 선정하는데 이용된다.
- 관리적 판단의 결정을 지원하고 구현과 유지보수에 내재하는 위험요소들을 이해하는데 이용된다.

소프트웨어 아키텍처 모델링은 아직 그 정도로 성숙되지 못하였다. 아키텍처를 모델링하는 여러 방법 중 어느 것을 선택하느냐는 일부는 모델의 목적에 도 일부는 개인적 취향에 따른다. 각 방법은 장점과 단점이 있어서 어느 경우에나 가장 적합한 보편적인 기법은 없다. 어떤 개발자는 클래스보다는 서브시스템을 강조하는 아키텍처를 표현하기 위해 UML(Unified modeling language)의 클래스 다이어그램을 사용한다. 보다 일반적으로 여러 유형의 상자와 화살표를 설명하는 범례가 첨부되기도 하는 단순한 상자와 화살표 다이어그램을 사용하여 소프트웨어 아키텍처는 모델링된다[3].

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].

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

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

아키텍처에서는 여러 가지 형태로 연동이 필요하다. 그 형태로는 아키텍처간이 될 수도 있고 아키텍처에서 연동되는 소프트웨어가 될 수도 있다. 아키텍처는 실세계의 요구사항을 수용하여 여러 가지 형태로 존재하게 되며, 많은 형태에 적합한 소프트웨어의 연동이 필요하게 된다. 이와 같은 과정에서 아키텍처와 소프트웨어간의 연동이 되는 경우, 연동이 가능한지의 확인이 선행되어야 한다. 아키텍처와 소프트웨어간의 연동을 위하여 인터페이스 연동, 데이터 베이스의 호환성의 관점으로 제시를 하였다.

3.1 인터페이스 연동

아키텍처의 설계는 계층간의 형태를 결정짓고 그 계층을 활용하여 데이터를 사용하게 된다. 그 결과 아키텍처를 활용하여 소프트웨어가 수행되는 형태가 된다. 이와 같은 수행이 이루어지기 위하여 소프트웨어가 아키텍처와 연동이 가능해야 한다. 본 절에서는 아키텍처와 소프트웨어 연동을 확인하기 위하여 인터페이스 연동에 초점을 두었다.

인터페이스는 모든 시스템에 존재하는 연동 또는 수행의 수단으로 활용된다. 인터페이스는 재사용을 위한 컴포넌트들 사이에서 사용되는 수단으로서 소프트웨어의 구동 수단으로 활용되고 있다. 인터페이스에는 속성과 오퍼레이션이 된다. 또한 고객의 요구사항을 추출하고 명세하는 과정에서 생성되는 결과이다.

인터페이스 연동 관점은 파라미터 데이터 형태, 형 변환 허용 여부, 형 변환 개수 또는 길이, 파라미터 개수를 중심으로 제안하였다.

아키텍처와 소프트웨어간의 연동을 확인하기 위한 기준은 Table 1과 같다.

Table 1. Check Point of Elements View Point of Interface Interoperability

Elements	Contents
Parameter data type	Check the parameter data type
Type conversion	Can be type conversion
Number or length of type conversion	Check the number or length of type conversion
Number of parameter	Check the number of parameter
Traceability	Check the traceability of data type

- 파라미터 형태

아키텍처와 소프트웨어간의 인터페이스에서 파라미터 형태를 확인한다. 파라미터 형태는 주로 데이터 형태와 연관되어 확인하게 된다. 데이터 형태에서는 소프트웨어를 구현할 언어에서 제공하는 데이터 형태와 아키텍처에서 제공되는 형태가 서로 연동이 될 수 있는가에 중점을 두고 연동 가능성에 대하여 확인을 하게 된다.

- 형 변환 허용 여부

연동이 불가능한 경우에는 데이터 형태 변화를 하여 연동을 할 수 있는 가능성에 대하여 확인을 해야 한다. 이를 위하여 데이터 형 변환이 소프트웨어와 아키텍처간에 적용을 할 수 있는지 확인해야 한다. 형 변환을 지원하는 범위의 확인을 통하여 데이터의 연동을 가능하게 한다. 형 변환이 지원되지 않는다면 데이터 연동이 제한되게 된다.

- 형 변환 개수 또는 길이

형 변환이 가능한 경우 형 변환의 대상과 개수, 길이에 대하여 확인을 해야 한다. 형 변환 대상의 식별을 통하여 데이터 연동에 대한 신뢰성을 향상시킬 수 있다. 또한 형 변환의 추적성을 유지하여 이전 형태에 대하여 저장을 하여야 한다. 형 변환이 일시적일수도 있고 영구적일수도 있어서 다른 아키텍처나 소프트웨어와 연동되는 경우에 이전 형태가 필요할 수 있게 된다. 이때 소프트웨어와 아키텍처의 형 변환에서 형 변환이 되는 대상은 형 변환의 개수가 적은 곳에서 수행되어야 변환에 의한 효율성을 높일 수 있다.

- 파라미터 개수

파라미터 개수는 소프트웨어와 아키텍처간의 데이터 교환에서 일치성을 확인할 수 있는 요소가 된다. 많은 데이터 교환에서 파라미터에게 전달되는 데이터의 불일치에서 오류가 발생하게 된다. 파라미터의 개수 확인은 실제 데이터 교환이 이루어지기 전에 수행되어야 한다. 파라미터 개수의 확인으로 데이터 교환의 오류를 미리 확인할 수 있다.

- 추적성

형 변환은 전반적으로 추적성을 전제로 한다. 이전에 존재했던 형태를 저장하여 형 변환의 변화 과정을 확인할 수 있어야 한다. 추적성을 유지함으로써 형 변환에서 발생하는 결함의 원인을 식별하여 수정할 수 있게 된다.

3.2 데이터베이스의 호환성

본 논문에서의 원시 데이터는 소프트웨어와 아키텍처가 요구분석과 설계 과정을 수행하면서 결정되게 되는 요소가 갖는 특성의 데이터를 의미한다. 아키텍처와 소프트웨어간의 호환성 중에서 데이터베이스가 가지고 있는 필드, 레코드의 특성을 확인해야 한다. 필드의 경우, 같은 정수형 데이터 형태라도 할당된 비트 수에 의하여 호환이 불가능하게 된다. 또한 레코드의 경우도 이질적인 데이터의 집합이므로 각 요소마다 연관되는 대상과 일치하지 않으면 호환성에 문제가 발생된다. 더 나아가서 레코드의 집합인 프로젝트의 경우에도 일치성 문제가 발생한다. 이와 같은 일치성은 호환성의 문제로 발생되어서 전체 시스템에 영향을 미치게 된다.

데이터베이스 호환성 관점의 확인 사항들은 필드 특성, 레코드 특성, 프로젝트 특성, 무결성으로 제시하였다. 데이터베이스 호환성 관점의 확인 사항들은 Table 2와 같다.

Table 2. Check Point of Elements View Point of Database Interoperability

Elements	Contents
Character of field	Check the character of field
Character of record	Check the character of record
Character of project	Check the character of project
Integrity	Check the integrity

- 필드 특성

데이터베이스에서 필드 특성은 요구사항 추출과정부터 가지고 있는 본질적인 특성을 의미한다. 같은 정수형 데이터를 선언한다고 해도 프로그래밍에서는 플랫폼과 언어의 특성에 제한적이 되지만 데이터베이스에서는 필드에 별도로 크기를 선언하게 되므로 이와 같은 특성의 확인은 매우 중요한 요인이 된다. 따라서 같은 데이터 형태의 경우에 크기가 달라서 호환성에 문제가 될 수 있다. 필드 특성을 확인하여 사전에 결함을 예방할 수 있게 된다.

- 레코드 특성

레코드는 이질적인 데이터 형태의 집합이 될 수 있다. 같은 종류의 필드로 구성될 수도 있지만 이질적인 데이터 형태로 구성되는 경우가 더 많다. 레코드 특성에서는 반복적이고 이질적인 필드들의 공통성을 발견하여 하나의 클러스터 화하여 조작과 데이터 연동의 효율성과 편의성을 제공할 수 있게 된다. 컴포넌트와 같은 재사용 가능한 소프트웨어를 연동 과정에서 생성하여 재사용성을 높일 수 있다. 이때 컴포넌트의 단위는 연동되는 소프트웨어와 아키텍처의 형태에 따라서 달라질 수 있다.

- 프로젝트 특성

아키텍처와 소프트웨어의 연동되는 대상이 프로젝트가 되는 경우이다. 이때 아키텍처와 소프트웨어간에는 같은 영역의 솔루션을 해결하는 경우에 프로젝트를 연동하게 된다. 프로젝트 특성에서는 연동대상과 함께 인터페이스 연동관점의 확인 사항들을 동시에 확인해야 한다.

- 무결성

무결성의 경우에는 아키텍처와 소프트웨어 연동과정에서 기존의 데이터의 내용을 원래의 내용에 변화 없이 전달할 수 있다는 것을 확인하는 것이다. 무결성은 필드, 레코드, 프로젝트 특성에 모두 적용이 된다.

본 장에서는 소프트웨어 연동을 위한 아키텍처간의 구성요소 확인을 위하여 인터페이스 연동과 데이터베이스 호환성 관점에서 확인사항들을 제시하였다. 각 확인 사항들은 체크리스트 형태로 활용하게 된다.

4. 적용 및 사례

4.1 적용 사례

3장에서 제시한 이론을 기반으로 사례에 적용하였다.

적용한 사례에서 아키텍처와 소프트웨어 연동 시 3장의 이론을 기반으로 확인하였다.

Table 3. Incase of Elements View Point of Interface Interoperability

Elements	Contents
Parameter data type	Integer, real, character, numeric
Type conversion	Can be change
Number or length of type conversion	Integer : 8bit Real : 16bit Character : 2bit numeric : 4bit
Number of parameter	number of parameter : 4
Traceability	Make the traceable table

Table 4. Incase of Elements View Point of Database Interoperability

Elements	Contents
Character of field	Can be the four field character : integer, real, character, numeric
Character of record	Can be the four field character : integer, real, character, numeric
Character of project	Can be the two field character : input, output
Integrity	Keep the integrity and use the traceble table

적용한 사례의 영역은 기상청에서 기상정보를 활용하여 서비스를 제공하는 영역이다. 연구의 특성상 아키텍처에서 추출한 내용을 기반으로 구현 내용을 제시하였다.

- JAVA로 XML파싱

```
import java.io.IOException;

import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.SwingConstants;
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.ParserConfigurationException;

import org.w3c.dom.Document;
import org.w3c.dom.Node;
import org.w3c.dom.NodeList;
import org.xml.sax.SAXException;

public class parsing{
    /**
     * @param args
     * @throws IOException
     * @throws SAXException
     * @throws ParserConfigurationException
     */

    public static void main(String[] args) throws SAXException, IOException, ParserConfigurationException {
        JFrame f = new JFrame();
        JPanel panel = new JPanel();
        f.add(panel);
        f.setSize(270,1000);
        f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        // DocumentBuilderFactory를 이용해서 DocumentBuilder 클래스 객체화
        DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
        DocumentBuilder builder = factory.newDocumentBuilder();
        Document doc = builder.parse("http://www.kma.go.kr/weather/forecast/mid-term-rss3.jsp?stnId=143"); //XML 파일 경로

        NodeList list = doc.getElementsByTagName("location");   // 노드리스트 0번 획득(첫번째 = location)
```

Fig. 1. Parsing from JAVA to XML 1

```
37        for(int i=0; i<list.getLength(); i++){
38            for(Node node = list.item(i).getFirstChild(); node!=null; node=node.getNextSibling()){
39                if (node.getNodeName().equals("city")){                    //노드이름이 city와 같다면 해당 도시 이름 표시
40                    JLabel label1 = new JLabel("====="+node.getTextContent()+"=====");
41                    label1.setHorizontalAlignment(JLabel.CENTER);
42                    panel.add(label1);
43                }
44                if(node.getNodeName().equals("data")){
45                    for(Node node2 = node.getFirstChild(); node2!=null; node2=node2.getNextSibling()){
46                        if (node2.getNodeName().equals("tmEf")){           //노드이름이 tmEf와 같다면 예보될 날짜 표시
47                            JLabel label6 = new JLabel("<html><br></html>");
48                            panel.add(label6);
49                            JLabel label2 = new JLabel(node2.getTextContent());
50                            panel.add(label2);
51                        }
52                        else if (node2.getNodeName().equals("wf")){        //노드이름이 wf와 같다면 구름의 정도 표시
53                            JLabel label3 = new JLabel(node2.getTextContent());
54                            panel.add(label3);
55                        }
56                        else if (node2.getNodeName().equals("tmn")){       //노드이름이 tmn와 같다면 최저 기온 표시
57                            JLabel label4 = new JLabel(node2.getTextContent());
58                            panel.add(label4);
59                        }
60                        else if (node2.getNodeName().equals("tmx")){       //노드이름이 tmx와 같다면 최고 기온 표시
61                            JLabel label5 = new JLabel(node2.getTextContent());
62                            panel.add(label5);
63                        }
64                    }
65                }
66            }
67        }
68        f.setVisible(true);
69    }
70
71 }
```

Fig. 2. Parsing from JAVA to XML 2

- JAVA로 XML파싱 결과

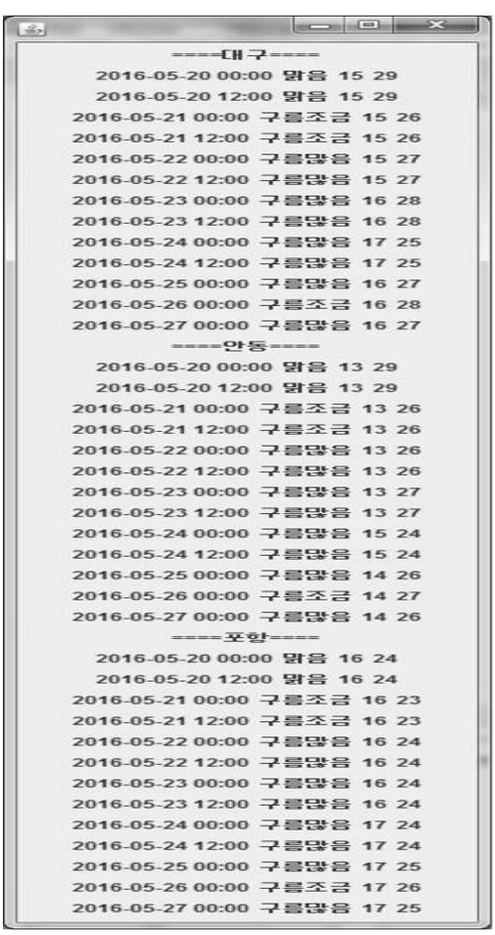

Fig. 3. Result of Parsing from JAVA to XML

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

4.2 검증

4.1절에서는 3장의 이론을 적용하기 위하여 사례를 제시하였다. 적용된 사례를 기반으로 이론을 검증하기 위하여 결함제거효율성 방법을 사용하였다. 결함제거효율성은 해당 개발 단계의 결함이 이후 단계의 개발 단계에서 결함이 발견되는 개수를 추출하였다. 그리고 해당 개발 단계 수행 중에 발견한 결함의 개수를 비교하여 결함제거효율성을 제시하였다. 따라서 3장의 이론을 적용하기 전과 적용한 후를 비교하여 제시하였다. 결과는 다음과 같다.

Table 5. Number of Defect (Requirements, Design)

	Ei	Ei+1
Requirement phase	10	15
Design phase	10	14

결함 개수를 기반으로 DRE를 산출하면 다음과 같다.
요구사항 분석 단계 : 10/10+15 = 0.4
설계 단계 : 10/10+14 = 0.416

다음은 3장의 이론을 적용한 후의 결함 개수를 산정하였다.

Table 6. Number of Defect (Requirements, Design)

	Ei	Ei+1
Requirement phase	15	8
Design phase	13	5

결함 개수를 기반으로 DRE를 산출하면 다음과 같다.
요구사항 분석 단계 : 15/15+8 = 0.652
설계 단계 : 13/13+5 = 0.722

결과를 분석하면 이론을 적용한 후의 결과가 적용 전보다 결함제거효율이 향상되었다. 특히 설계 단계의 결함이 많이 제거된 것을 확인할 수 있다.

5. 결 론

본 논문에서는 소프트웨어 연동을 위한 아키텍처간의 구성요소의 확인할 사항을 제시하고 사례에 적용하여 효율성을 검증하였다. 아키텍처와 소프트웨어간의 연동을 위하여 인터페이스 연동, 데이터 베이스의 호환성의 관점으로 제시를 하였다. 따라서 제안 내용을 활용하여 아키텍처와 소프트웨어 연동 시 발생할 수 있는 문제를 예방하고자 하였다. 향후 연구로는 아키텍처와 소프트웨어 연동을 위한 도구를 개발하고자 한다.

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] Garlan david, “Software architecture: A road map,” In anthony finkelstein (ed.), *The future of software engineering*, New york: ACM press, 2000.
[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] Yu chuljung, “Software engineering theory and practice,” Hansan, 2013
[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.
[17] Kim Chisu, “Software engineering,” Hanbit academy, 2015.

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)