소프트웨어 아키텍처 설계의 근본 원리들

(Fundamental Principles for Software Architecture Design)

강성원*

(Sungwon Kang)

요 약 이 논문에서는 소프트웨어의 개념과 소프트웨어 개발의 중요성을 알아본 뒤, 효과적인 소프트웨어 개발을 위한 필수적인 기술로서 열 두 개의 근본적인 소프트웨어 아키텍처 설계원리를 제시한다. 아키텍처 설계단계들을 문제의 파악, 아키텍처 모델링 방법의 결정, 아키텍처 설계의 수행 그리고 설계된 아키텍처의 평가의 네 개의 단계 군으로 나누고, 이 단계군 안에서 그리고 단계군 간에 활용되는 아키텍처 설계의 원리들을 식별하고 그 역할을 설명한다.

키워드 소프트웨어 아키텍처, 소프트웨어 아키텍처 설계, 소프트웨어 아키텍처 설계원리, 소프트웨어 아키텍처 설계절차

Abstract This paper first examines the notion of software and the importance of software development and then presents twelve fundamental principles for software architecture design as the key enabling technology for effective software development. This paper divides design steps into four groups, i.e. analyzing the problem, deciding architecture modeling methods, architecture design process and architecture evaluation. Then it identifies the principles within and across the various steps of software architecture design and explains their roles.

Key words Software architecture, software architecture design, software architecture design principles, software architecture design procedure

1. 컴퓨터 소프트웨어

1.1 소프트웨어와 컴퓨터 소프트웨어

소프트웨어라는 용어는 컴퓨터 하드웨어에 대비되는 표현으로서, 컴퓨터를 위한 프로그램을 지칭하는 용어로 사용된다. 컴퓨터 소프트웨어는 처음에는 컴퓨터를 동작하게 하는 일련의 컴퓨터 명령을 의미하였다. 그러나 프로그래밍 언어, 프로그램 라이브러리, 프로그램 프레임워크, 미들웨어 등의 발전으로 컴퓨터와의 세부적인 연결을 직접 표현하지 않고도 사용자가 필요한 연산, 데이터 처리, 활동 등을 표현할 수 있게 되면서, 컴퓨터를 제어하는 용도보다는 인간생활의 니즈(needs)를 표현하는 용도로 사용되는 방향으로 발전되어 가고 있다.

따라서 컴퓨터를 명령하기 위한 표현으로서의 소프트웨어를 넘어서, 소프트웨어가 무엇이며, 무엇을 위한 것인지를 생각해 볼 필요성이 자연스럽게 대두 되었다. 그것은 컴퓨터 소프트웨어의 역할이 본질적으로 무엇인가를 생각하며, 현대적인 컴퓨터가 존재하기 이전의 세계로 그 역할을 투영해봄으로써 더 잘 생각해 볼 수 있을 것이다.

1.2 역사 속의 소프트웨어 개념

인류는 오랫동안 천을 짜는 기계인 방직기를 이용하여 다양한 무늬를 갖는 천을 만들어 왔다.

* 종신회원 : 한국과학기술원 전산학과 부교수
sungwon.kang@kaist.ac.kr
논문접수 : 2010년 11월 20일
심사완료 : 2010년 12월 23일

이 때 다양한 무늬를 만들기 위하여 먼저 카드에 방직기의 동작을 제어할 수 있는 구멍을 무늬에 맞게 뚫는데, 이 때 방직기는 하나의 하드웨어이고 카드의 구멍의 배열은 하나의 소프트웨어로 방직기를 위한 프로그램인 것이다.[1] Ada Augusta (1815 – 1852)는 Charles Babbage (1791 – 1871)의 해석엔진(Analytical Engine)[2]을 위한 프로그램을 설계하였는데, 이 프로그램을 최초의 컴퓨터 소프트웨어로 간주하기도 한다. 또한 초기의 컴퓨터에는 전기-기계식 컴퓨터[3]도 있었으며 프로그래머가 스위치 보드 하드웨어에 직접 프로그래밍을 하기도 하였다.

이와 같이 현대적 컴퓨터가 존재하기 이전에도 소프트웨어는 존재하였고 사람들은 프로그래밍 활동을 하였으며, 현대적 컴퓨터가 등장한 후에 새로운 국면을 맞게 되지만, 소프트웨어와 프로그래밍은 이런 긴 역사를 가지고 있다고 볼 수 있다.

1.3 서비스, 소프트웨어 서비스, 컴퓨터 소프트웨어 서비스

컴퓨터 소프트웨어는 하나 하나의 단계가 엄밀히 수행될 수 있는 알고리즘으로서의 특징을 통하여 우리가 필요로 하는 계산, 정보의 전달, 물품의 구매, 전통적인 제반 분야의 서비스 등의 니즈를 달성하는 서비스 매체로서의 특징을 갖는다. 거꾸로, 컴퓨터를 사용하지 않더라고, 그러한 니즈를 달성하게 해주는 엄밀한 절차가 존재한다면 그것을 소프트웨어라고 부를 수 있을 것이다. 과거에는 오늘날 컴퓨터 소프트웨어가 하는 많은 서비스를 컴퓨터 소프트웨어 없이 인간의 두뇌를 통하여 달성하였던 것이다.

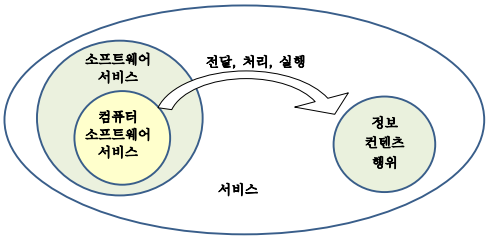

그림 1. 서비스, 소프트웨어 서비스, 컴퓨터 소프트웨어 서비스의 개념

그림 1은 전통적인 서비스의 개념과 소프트웨어 서비스 그리고 컴퓨터 소프트웨어 서비스의 개념 간의 관계를 보여 준다. 전통적인 산업의 분류상 1차, 2차 산업이 각각 원료의 수집과 원료의 가공을 통한 유형적 물건의 조달 및 제조활동이었던 비하여, 무형적인 대상으로 사용자의 니즈를 충족시켜주는 행위가 3차 산업인 서비스 산업으로 분류되었다. 소프트웨어 서비스는 정보와 컨텐츠를 제작, 전달, 가공하는 활동을 포함한다. 또 서비스는 우리가 소프트웨어 서비스라고 부르는 것과 아울러 정보와 컨텐츠를 포함한다. 정보와 컨텐츠는 디지털화 되면 컴퓨터 소프트웨어 서비스가 가능한 형태가 되어, 전달, 처리 등의 컴퓨터 소프트웨어 서비스를 통하여 가공되어 부가가치를 창출하는 새로운 것이 될 수도 있다.[4]

1.4 컴퓨터 소프트웨어의 역할의 발전

소프트웨어라는 용어가 컴퓨터 소프트웨어라는 용어를 통하여 일상화 된 지는 오래되지 않지만, 소프트웨어가 인간 사회에 등장한 것은 매우 오래 전이었다. 그러나 컴퓨터의 등장으로 인하여 인간이 소프트웨어를 이용하는 방법은 극적으로 넓어졌고, 이제는 컴퓨터 소프트웨어는 산업과 생활을 위하여 업무, 공장, 오락, 교육 등 광범위한 분야에서 필수적인 기능을 수행하게 되었고 나아가 과학기술의

1) Joseph Jacquard(1752-1834)는 1801년 이렇게 동작하는 자동 방직기를 설계하였다.
2) Charles Babbage는 결국 해석엔진을 완성하지 못하였다.
3) 1940년대 Konrad Zuse (1910-1995)는 최초의 전기-기계식 컴퓨터 Z3를 만들었다.
4) 이와 같은 통찰은 다시, 어떤 전통적인 산업분야나 생활분야가 컴퓨터 소프트웨어의 도움을 받아 발전될 수 있는가의 방향을 시사할 것이다.

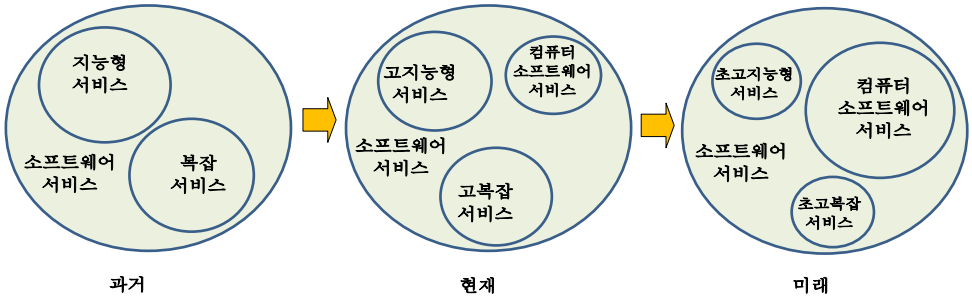

그림 2. 컴퓨터 소프트웨어의 역할의 발전

발전을 이끄는 도구로서도 눈부신 역할을 하게 되었다. 그림 2은 이런 컴퓨터 소프트웨어 역할의 발전 과정을 보여준다.

소프트웨어 서비스 중에서 단순하고 간단한 것들은 컴퓨터의 등장과 함께 바로 현재에 컴퓨터 소프트웨어 서비스로 바뀌어, 높은 지능을 요구하거나 복잡한 서비스가 아닌 것은 현재에는 컴퓨터 소프트웨어 서비스로 제공되게 되었다. 가까운 미래에는 매우 복잡하거나, 매우 높은 지능을 요구하는 서비스들을 제외하고 컴퓨터 소프트웨어 서비스에 의하여 제공될 것이면, 이 들 중에 많은 것들도 궁극적으로 컴퓨터 소프트웨어 서비스로 제공되게 될 것이다.

1.5 컴퓨터 소프트웨어 기술의 중요성

컴퓨터는 컴퓨터 소프트웨어를 이용하기 위한 장치이고 컴퓨터 소프트웨어를 이용 가능하게 만들어 주는 도구이다. 과거에 석기시대에는 돌을 잘 다루는 부족이, 철기시대에는 철을 잘 다룰 줄 아는 국가가, 산업혁명의 시대에는 기계를 잘 만들어 다룰 줄 아는 민족이 세계를 지배했던 것처럼 컴퓨터 시대에 컴퓨터 소프트웨어를 잘 만들어 다루는 사람들이 세상을 지배하게 될 것이라고 보아도 과언이 아닐 것이다.[5] 이와 같은 추측을 뒷받침하는 증거로, 현대의 전쟁이 로봇병사와 무인비행기를 이용하여 수행되고 있고, 현대적 기업이 비즈니스를 컴퓨터 소프트웨어로 수행하여 비용을 절감하고 고객의 취향을 실시간에 파악함으로써 기업이 경쟁력을 갖게 되는 등의 예에서 볼 수 있다. 그리고 컴퓨터소프트웨어를 잘 만드는 방법에 대한 연구도 당연히 컴퓨터의 역사만큼이나 길었던 것이다.

Mark Weiser는 수 많은 컴퓨터가 우리 생활 속에 밀집되어 퍼져있는(pervasive) 새로운 문명 세계에 대한 비전을 제시하였다 [9]. 이 비전은 이미 현실로 되어 가고 있다. 이러한 현상은 이 많은 컴퓨터들이 원하는 대로 기능하도록 하기 위한 많은 다양한 컴퓨터 소프트웨어가 필요하다는 것을 의미하고, 나아가 이들 컴퓨터를 이용한 다양한 (소프트웨어) 서비스들도 만들어져야 한다는 것을 의미한다.

1.6 컴퓨터 소프트웨어를 어떻게 잘 다룰 수 있는가?

좋은 소프트웨어를 만들기 위하여, 우선 정확한 요구사항의 명세가 있어야 한다. 올바른 명세를 확보한 후에는 물론 그것을 잘 설계하고 잘 구현하여야 한다. 또한 시험을 철저하게 하여야 할 뿐 아니라, 이 전체를 위한 공정이 잘 계획되고 준수되어야 한다. 그러나 좋은 소프트웨어가 만들어지기 위해서는 무엇보다도 설계가 잘 만들어져야

5) C++프로그래밍 언어의 설계자인 Bjarne Stroustrup는 "Our civilization runs on software."는 말을 하여 이를 잘 요약해 주고 있다. (http://www.handbookofsoftwarearchitecture.com/index.jsp?page=main&part=Preface 에서 재인용)

한다. 예를 들어 시험이나 공정이 아무리 효과적이어도, 잘못된 구현이나 설계의 문제점을 지적해 줄 뿐이고, 올바른 구현과 설계를 만드는 작업은 다시 수행되어야 하기 때문이다. 또한 구현과 설계 가운데 구현이 설계에 의존하기 때문에 구현에 문제가 있을 때보다 설계에 문제가 있을 경우 이를 바로 잡는데 훨씬 더 많은 비용이 들어가게 된다.

어느 정도의 규모를 가진 소프트웨어의 경우 반드시 설계를 거쳐서 구현을 해야 한다는 것은 잘 알려진 사실이다. 만들어야 하는 소프트웨어의 규모가 커지고 복잡도가 높아짐에 따라, 이 두 가지를 정면으로 다룰 수 있는 방법이 필요하게 되었다. 이를 위하여 설계는 다시 아키텍처 설계와 상세 설계로 나뉘어져 아키텍처 설계단계에서 이 문제들의 해결을 시도하게 된다.

2. 소프트웨어 아키텍처의 개념과 중요성

1969 NATO Conference on Software Engineering Techniques 에서 Ian P. Sharp는 다음과 같이 말하였다:

"우리가 필요한 것은 소프트웨어공학 이외에도 더 있다. 그것에 관하여 우리가 조용히 얘기하여 왔지만, 공개적으로 논의하고, 관심을 가져야 할 것이다. 그것은 바로 소프트웨어 아키텍처이다. 아키텍처는 공학과 다르다. 이 말의 의미를 생각해 보기 위하여 예를 들어 OS/360 운영체제의 예를 들어 보자. OS/360은 각 부분별로는 매우 코딩이 잘 되어 있다. 세부적으로 들어가 보면 OS의 부분 부분은 우리가 좋은 프로그래밍 기법이라고 보는 기법들과 모든 개념들을 쓰고 있다. 그런데 OS/360가 형체 없는 프로그램의 덩어리(amorphous lump of program)가 된 이유는 아키텍트가 없었기 때문이다. 설계는 많은 그룹의 엔지니어들에게 위임되었고, 그들은 각자의 아키텍처를 반영하여야 하였다. 그리고 이 덩어리들을 최종적으로 통합하였을 때에는 전체가 부드럽고 아름다운 소프트웨어가 되지 못하였다." [7]

2.1 전통적 설계 접근 방법의 불충분성

Ian P. Sharp의 말은 소프트웨어 개발에 있어서 아키텍처가 어떤 역할을 수행해야 하는지를 잘 보여준다. 전통적으로 소프트웨어는 그림 3이 보여주는 것처럼 요구사항분석, 설계, 구현, 통합 및 시험이라는 단계를 거쳐 개발되었다. 그러나 전통적 소프트웨어 개발에서의 설계의 역할은 오늘날 우리가 아키텍처에 기대하는 역할에 크게 미치지 못하였다. 예를 들어, 객체지향설계 방법의 경우 시스템을 많은 객체들의 구성으로 보라고 하지만, 많은 객체들을 쉽게 이해하고 다룰 수 있도록 어떻게 더 큰 단위로 볼 수 있는가에 대하여는 말해 주고 있지 않다.

그림 3. 개발순기

아키텍처 설계의 목적이 요구사항과 구현을 "연결" 시키는 것인데, 전통적인 설계방법들도 요구사항과 구현의 간격을 메워 이들을 연결시키는 역할을 수행한다. 그러나 이 둘은 그 접근방식에서 다르다. 그림 4에서 네모는 산출물 혹은 모델을 나타내고 화살표는 자료의 흐름을 나타낸다. 그림 4(a)처럼 설계나 아키텍처설계 없이는 구현자는 즉흥적인 방법으로 구현을 하게 된다. 전통적인 설계방법들은 요구사항들을 구현으로 가져가는 길을 제시한다(그림 4(b)). 반면 아키텍처 설계는

요구사항으로부터 설계로 가는 기계적인 단계들을 명시하려고 하는 대신, 서로 다른 대안 아키텍처가 이 들에 기초한 선택과 같은 문제에 관심을 둔다 (그림 4(c)). 따라서 전통적인 설계와 아키텍처 설계는 상호보완적이다.

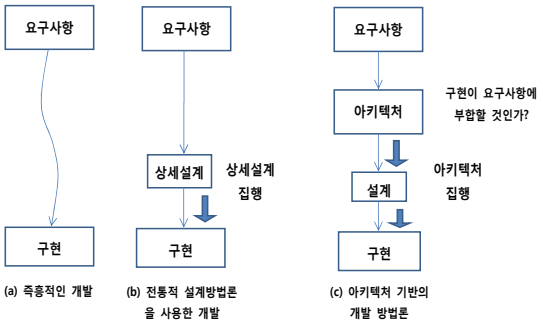

그림 4. 아키텍처 기반 개발의 장점

2.2 소프트웨어 아키텍처의 역할

따라서 M. Shaw, D. Garlan, P. Kruchten, Perry을 비롯한 소프트웨어 아키텍처 연구의 선구자들은 소프트웨어 개발의 중추적인 역할(pivotal role)을 수행하는 아키텍처의 필요성을 역설하였다.

중심축(pivot)이 한 방향으로부터 온 힘을 다 받아서, 다른 방향으로 전달하는 역할을 수행한다 (그림 5 참조).

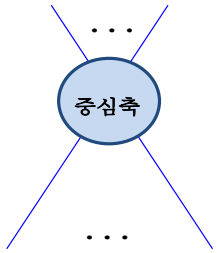

그림 5. 중심축의 개념

소프트웨어 개발에서 아키텍처도 바로 이러한 역할을 수행한다. 따라서 그림 6이 표현하고 있는 것처럼, 아키텍처 설계는 요구사항 분석 활동으로 얻어진 요구사항을 다 충족시킬 수 있는 시스템이 궁극적으로 만들어 질 수 있도록 하기 위한 설계 활동으로서 설계 및 구현을 위한 구조적(structural) 및 비구조적(nonstructural) 틀을 제공하는 것이다. 여기서 '틀'은 아키텍처 설계의 결과물로서 집행을 요구하는 결정 혹은 모델을 말한다. 따라서 구조적 틀이라 함은 시스템을 컴포넌트로 구성된 구조로 보아 시스템 개발문제를 용이하게 끌어가도록 결정된 컴포넌트의 구조모델을 말하고, 비구조적 틀은 이 구조모델 이외의 다른 아키텍처설계의 결정들을 말한다.

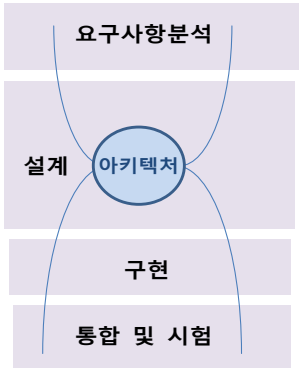

그림 6. 소프트웨어 개발의 중심축으로서의 아키텍처

3. 소프트웨어 아키텍처의 정의

이 논문의 제 1절에서는 소프트웨어 아키텍처의 기본적인 개념을 살펴보았다. 이 절에서는 소프트웨어 아키텍처에 대한 엄밀한 정의를 내려, 이어지는 소프트웨어 아키텍처 설계에 대한 논의 의 기반을 확립한다.

3.1 아키텍처 와 소프트웨어 아키텍처

1980년대까지 아키텍처라는 용어는 컴퓨터 아키텍처의 의미로 쓰였고, Fred Brooks, Butler Lampson, David Parnas, John Mills등을 통하여 시스템의 "소프트웨어 조직"의 의미로도 쓰이기 시작하여, 소프트웨어 아키텍처가 본격적으로 연구되기 시작하였다.

3.2 설계의 다양한 수준

건물, 교량, 도시를 건설할 때 건축가는 먼저 그 설계를 한다. 설계는 중요한 결정들을 하기 위한 아키텍처설계와 세부작업의 바탕이 되는 상세설계로 나눌 수 있다. 예를 들어 교량의 아키텍처설계에는 강물의 속도와 강의 폭, 단위시간에 교량을 건너는 차량의 평균수 및 최대수, 차의 최대 무게 등을 고려하여 다리의 공법, 구조, 폭 과 강도가 결정되어야 할 것이다. 교량의 상세 설계에서는 교량의 아키텍처 설계의 결정들이나 다른 상세설계 작업에 큰 파급효과가 없는 사항들이 결정된다.

이와 같이 소프트웨어 개발에서도 다양한 수준의 설계작업이 있고, 소프트웨어 시스템의 아키텍처 설계에 제약을 주는 상위설계로서의 시스템 아키텍처의 설계 가 있고 소프트웨어 아키텍처의 제약 안에서, 세부적인 설계작업을 수행하게 되는 상세설계가 있다. 그림 7은 이 세 개념들 사이의 관계를 보여준다.

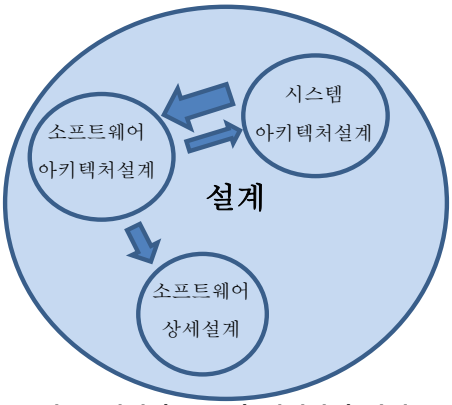

그림 7. 설계의 종류와 설계간의 관계

이들은 일반적인 설계활동의 하나로서 설계가 갖는 일반적인 특성을 공통적으로 갖는다. 그러나 또한 이들은 각기 다른 수준의 설계를 지칭하여, 시스템 아키텍처는 하드웨어와 소프트웨어로 구성된 시스템의 아키텍처를 지칭하며 소프트웨어 아키텍처의 결정을 제약하게 되고, 소프트웨어 아키텍처 설계는 소프트웨어 상세설계의 결정을 제약하게 된다. 그림 7에서 화살표의 굵기는 영향력의 크기를 나타낸다.

3.3 과거 연구자들의 소프트웨어 아키텍처 정의

과거 많은 소프트웨어 아키텍처의 연구자들이 소프트웨어 아키텍처에 대하여 다양한 정의를 내렸다. 그 중 다음과 같은 대표적인 정의들이 있다:

"소프트웨어 아키텍처는 소프트웨어 시스템들의 큰 규모의 구조와 실행에 관한 연구이다"[7].

소프트웨어 아키텍처란 "시스템의 근본적인 조직형태로써, 그것은 구성컴포넌트들과, 그들 서로와 환경에 대한 관계 그리고 그 설계와 진화를 관장하는 원칙들에 담겨있다" [4].[6]

소프트웨어 "아키텍처란 1)시스템의 구조를 나타내는 구조적 구성요소와 그들을 결합시키는 인터페이스, 2)그들의 협동을 통해 나타나는 구조적 구성요소와 행위적 구성요소들의 결합을 점진적으로 서브시스템에 맵핑을 하는 것, 3)구조를 이끌어 나가는 아키텍처 스타일의 선택"에 관한 중요한 결정들의 집합이다. [5][8].[7]

"프로그램 혹은 컴퓨팅 시스템의 소프트웨어 아키텍처란 소프트웨어 구성요소, 이들 구성요소의 가시적인 속성, 그리고 구성요소 사이에 관계로

6) Software architecture is "the fundamental organization of a system, embodied in its components, their relationships to each other and the environment, and the principles governing its design and evolution" [6].

7) "An architecture is the set of significant decisions about the organization of a software system, the selection of the structural elements and their interfaces by which the system is composed, together with their behavior as specified in the collaborations among those elements, the composition of these structural and behavioral elements into progressively larger subsystems, and the architectural style that guides this organization – these elements and their interfaces, their collaborations, and their composition" [7,8].

구성된 시스템의 전체적인 구조 또는 구조들을 말한다"[1].[8]

3.4 소프트웨어 아키텍처의 정의

소프트웨어 아키텍처는 소프트웨어 시스템이 존재하기 이전에 그것에 관한 관한 많은 중요한 결정을 내릴 수 있게 하고 아키텍처 수준에서 소프트웨어 시스템의 분석을 가능하게 함으로써 계획된 소프트웨어 개발이 가능하도록 한다. 그림 8는 소프트웨어 아키텍처가 아키텍처 설계를 위하여 필요한 형태로 요구사항이 철저히 분석될 수 있도록 가이드 하는 한편, 아키텍처설계 단계에서 내려진 소프트웨어 구조를 포함한 제반 아키텍처적인 결정들이 상세설계, 구현, 통합 및 시험에까지 집행될 수 있도록 가이드 하는 구체적인 산출물임을 보여준다.

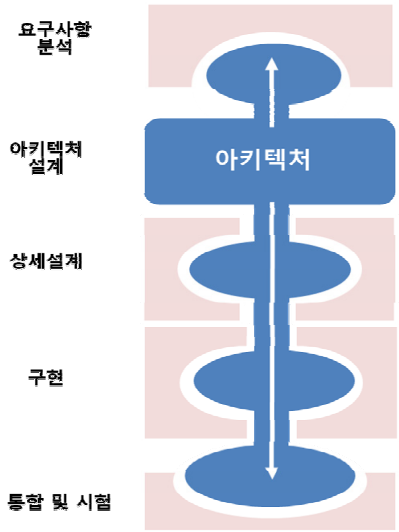

그림 8. 개발순기에 있어서 아키텍처의 역할

8) 이 정의는 논문 [4]에서 1994년에 SEI에서 논의되었다고 하는 "프로그램/시스템의 컴포넌트들이 갖는 구조, 그들의 관계 그리고 그들의 설계와 시간이 흐르면서 발생하는 진화를 지배하는 원칙과 가이드라인 (The structure of the components of a program/system, their interrelationships, and principles and guidelines governing their design and evolution over time)" 이라는 정의와 유사한 점이 있다.

소프트웨어 시스템은 개발 이후에도 많은 시간과 비용을 들여 진화하게 되는데, 아키텍처는 소프트웨어 진화과정에까지도 그 영향을 미친다. 따라서 우리는 다음과 같이 소프트웨어 아키텍처를 정의한다.

> **소프트웨어 아키텍처의 정의**
> 소프트웨어 아키텍처는 소프트웨어 시스템의 구조를 비롯한 시스템 개발에 중요한 영향을 미치는 결정들로, 소프트웨어 시스템 개발에서 특정 시스템에 대하여 요구되는 기능과 품질을 확보하고 또한 소프트웨어 시스템의 구축 및 지속적인 개선이 용이하도록 하는 역할을 한다.[9]

4. 소프트웨어 아키텍처의 설계

소프트웨어 아키텍처를 제 3절에서와 같이 정의한다면, 그러면 소프트웨어 아키텍처는 어떻게 만들 수 있을까, 즉 소프트웨어 아키텍처는 어떻게 설계하여야 하는가 하는 문제가 대두된다. 이 절에서는 소프트웨어 아키텍처 설계가 가져야 하는 본질적인 특징들을 살펴본다.[10]

4.1 분할정복으로서의 설계활동

분할 정복은 복잡한 문제를 풀기 위하여 원래의 문제를 작은 문제들로 나누고 먼저 작은 문제들에 대한 답을 얻은 뒤에 이 답들을 연결하여 전체 문제에 대한 답을 찾는 방법이다.

9) 이 정의는 논문 [4]에서 1994년에 SEI에서 논의되었다고 하는 "프로그램/시스템의 컴포넌트들이 갖는 구조, 그들의 관계 그리고 그들의 설계와 시간이 흐르면서 발생하는 진화를 지배하는 원칙과 가이드라인 (The structure of the components of a program/system, their interrelationships, and principles and guidelines governing their design and evolution over time)" 이라는 정의와 유사한 점이 있다.

10) 이러한 설계의 일반원리는 제 4절 그림 11 아키텍처 설계 절차에도 반영되어, 분할 정복원리는 컴포넌트와 커넥터의 이분법을 요구하고, 분석과 합성의 원리는 설계절차의 형태를 결정하고, 경험과 창의성의 결합원리는 설계에 적용되는 기법의 원리를 결정한다.

문제의 규모가 작고 복잡도가 낮을 경우에는 이런 접근 방법이 불필요할 수 있다. 그러나 문제의 규모가 클 경우 문제를 분할하여 작은 문제로 바꾸고 이들에 대한 해결을 통합하여 전체에 대한 해결하려 하는 것은 자연스럽고도 강력한 문제 해결 방법이다. 소프트웨어 아키텍처 설계는 본질적으로 규모와 복잡성을 다루는 접근방법이므로, 분할정복이 아키텍처 설계의 지배적인 해결 방식인 점은 아키텍처 설계를 일반적인 설계와 차별화해 주는 가장 큰 특징이다.

4.2 분석과 합성으로서의 설계활동

아키텍처 설계를 포함한 모든 설계는 창조적인 합성(synthesis)의 활동으로서[11] 좋은 결과를 가져오기 위하여는 분석(analysis)이라는 과정을 동반한다.[12] 그림 9는 통합과 분석이 서로 맞물리어 반복적으로 이어질 때, 좋은 설계결과가 만들어진다는 것을 나타낸다.[13] 아키텍처 설계도 하나의 설계 활동으로서 설계가 갖는 이러한 일반적인 특징을 갖는다.

그림 9. 합성과 분석의 반복으로 이루어진 과정으로서의 설계

4.3 경험과 창의성의 결합으로서의 설계활동

설계문제를 해결하기 위해서는 경험적인 수단과 창의적인 수단을 모두 동원하여야 한다. 앞에서 말한 분석과 합성은 일반적인 방법으로 원칙적으로 모든 문제에 대한 답을 줄 수 있는 틀이다. 그러나 합성과 분석에만 의존하는 것이 가장 효율적인 방법은 아니다. 경험적인 지식이 적용 가능할 때에는, 이를 잘 활용함으로써 많은 경우 짧은 시간에 더 높은 품질의 설계결과에 도달할 수 있다.

4.4 아키텍처 설계의 입력물과 출력물

설계가 분석과 합성의 반복적인 과정이라는 점에서 한 걸음 더 나아가서, 좀 더 상세히 설계의 과정을 들여다보기로 한다. 방법 혹은 과정은 그것이 어떤 것이든, 그것의 진행을 위하여 필요한 입력물이 있고,[14] 진행의 결과로서 만들어지는 출력물이 있다. 그렇다면 아키텍처 설계라는 활동의 입력물과 출력물은 무엇인가? 그림 10은 이를 도식적으로 보여준다.

소프트웨어 개발의 출발점은 요구사항이다. 아키텍처도 소프트웨어 개발을 체계적이고 목적에 맞게 달성할 수 있게 하는 역할을 수행하기 때문에 아키텍처 설계 역시 요구사항에서 출발하지만, 시스템 개발이 모든 요구사항을 그 달성대상으로 하는데 비하여, 아키텍처 설계는 아키텍처에 관련된 대표적인 요구사항들만이 주된 관심의 대상이 된다. 이러한 특별한 요구사항들을 아키텍처 드라이버라고 부른다. 한편 설계의 결과는 아키텍처를 문서화한 아키텍처 문서가 주된 출력물이고, 이에 대한 추가적인 관련사항을 정리한 아키텍처 가이드라인이 이차적인 출력물이다.

11) 또한 지적으로 "난이도가 높은 (challenging)" 활동이다.
12) 광범위한 의미의 설계는 이 둘을 포함하지만, 좁은 의미의 설계는 합성을 의미한다.
13) Kruchten은 논문 [10]에서 설계의 성격을 이와 같이 규정한다.
14) 입력물을 필요로 하지 않는 방법 혹은 과정은 그 다지 흥미롭지 못하다. 왜냐하면 그 결과가 항상 동일하거나, 달라질 경우에도 결과를 결정짓는데 그 방법 (또는 과정)의 이용자가 역할을 수행할 수 없기 때문이다. 시스템의 이용자는 유용한 방법 혹은 과정에서는 필요한 출력물이 나오도록 적절한 입력물을 제공하는 역할을 수행한다.

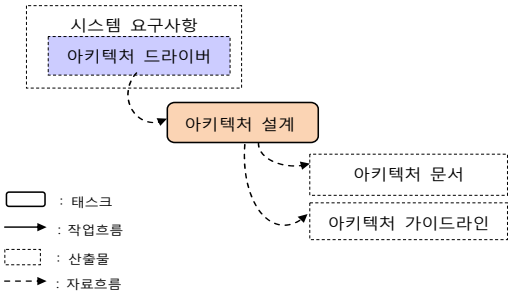

그림 10. 아키텍처 설계의 입력물과 출력물[15)]

5. 소프트웨어 아키텍처 설계의 근본 원리들

문제의 파악, 아키텍처 모델링 방법의 결정, 아키텍처 설계의 수행 그리고 설계된 아키텍처의 평가의 네 개의 단계 군으로 구성된 것으로 보고, 이들 단계군 안에서 그리고 단계군 간에 필요한 아키텍처 설계 원리들에 대하여 알아본다. 본 논문에서는 소프트웨어 아키텍처 설계에 필요한 근본적인 원리를 다음 12개로 본다:

1. 아키텍처 드라이버
2. 품질속성, 검증가능성, 품질속성시나리오
3. 문제분석
4. 컴포넌트와 커넥터
5. 아키텍처 스타일
6. 소프트웨어 아키텍처를 보는 관점체계
7. 설계의 일반원리
8. 아키텍처 설계 절차
9. 아키텍처 패턴
10. 품질속성 설계전술
11. 아키텍처의 분석
12. 아키텍처의 평가

15) 그림에서 "산출물(artifact)"이라 함은 소프트웨어 개발과정에 만들어지는 모든 결과물들을 총칭한다. "태스크(task)"는 한 명의 수행자가 수행하는 일을 지칭하며, 소프트웨어 설계 모델링의 표준적인 언어인 UML의 활동 다이어그램(activity diagram)은 더 이상 분해하지 않는 일의 단위를 태스크라 부르고 다른 태스크 혹은 활동으로 구성된 일의 단위를 활동이라고 부르고 있다. 그림의 다이어그램도 UML과 일관된 기호를 사용하고 있다.

이 12 개의 원리들이 아키텍처 설계 과정에 적용되는 단계를 그림 11은 보여준다. 그림 11은 또한 아키텍처 설계절차가 네 개의 단계군으로 구성되는 것으로 보여준다.

네 개의 단계군 가운데 첫 번째 단계군은 요구사항을 아키텍처 설계가 적용되기에 좋은 형태로 만들어 가는 원리에 대한 것이다. 두 번째 단계군은 아키텍처를 어떻게 표현할 것인가에 대한 원리로써 아키텍처를 보는 시각과 결과모델의 형태를 결정짓는다. 특히 컴포넌트와 커넥터의 이분법은, 아키텍처 스타일이나 아키텍처 관점과 같은 아키텍처의 다른 보다 구체적인 형태들의 공통적인 기반을 제시한다. 세 번째 단계군은 이 두 그룹의 원리를 적용한 결과를 바탕으로 하여 아키텍처 설계를 수행하는 원리들의 그룹이다. 이들은 일반적인 설계원리와 경험적인 지식을 이용하는 원리들의 두 가지로 구성된다. 이 세 번째 단계군은 전체 설계절차 중에서도 설계결과가 직접 결정되는 단계이므로 이 단계군을 좁은 의미의 아키텍처 설계라고 부르고, 이와 구별하기 위하여 전체설계절차를 넓은 의미의 아키텍처 설계라고 부르기로 한다. 네 번째 단계군에서는 아키텍처 평가가 수행되는데, 아키텍처 설계결과가 나오면 그에 대한 평가를 하여 아키텍처를 채택 또는 수정 결정을 하거나, 다른 대안을 찾는 결정을 내릴 수도 있게 된다.

이 절의 이하에서는 아키텍처 설계절차에 필요한 12개의 원리의 역할에 대하여 하나씩 설명한다.

5.1 아키텍처 드라이버

소프트웨어 시스템은 소프트웨어 시스템에 대한 요구사항으로부터 만들어진다. 소프트웨어 아키텍처는 소프트웨어 시스템의 아키텍처이기 때문에 아키텍처가 시스템에 대한 요구사항으로부터 설계된다는 것은 당연한 사실이다. 그러나 시스템에 대한 모든 요구사항이 아키텍처 설계를 위하여

필요하지는 않다. 시스템에 대한 요구사항들은 아키텍처에 영향을 주는 요구사항들과 그렇지 않은 요구사항들로 나뉘어 진다. 전자의 종류의 요구사항을 아키텍처 드라이버라고 부르는데, 따라서 아키텍처 설계에 더 잘 집중 할 수 있기 위하여는, 아키텍처 드라이버를 먼저 잘 가려낸 후 아키텍처 설계에 효과적으로 반영하여야 한다.

5.2 품질속성, 검증가능성, 품질속성 시나리오

아키텍처의 드라이버 중에서도 어떠한 결정요인들은 쉽게 아키텍처에 반영이 되는데 비하여, 어떤 결정요인들은 간단히 아키텍처에 반영될 수 없는 요인들이 있다. 예를 들어, '시스템이 계층적 구조를 가져야 한다'라는 제약사항이 있으면, 이 요구사항은 시스템의 아키텍처가 계층적 구조를 갖도록 함으

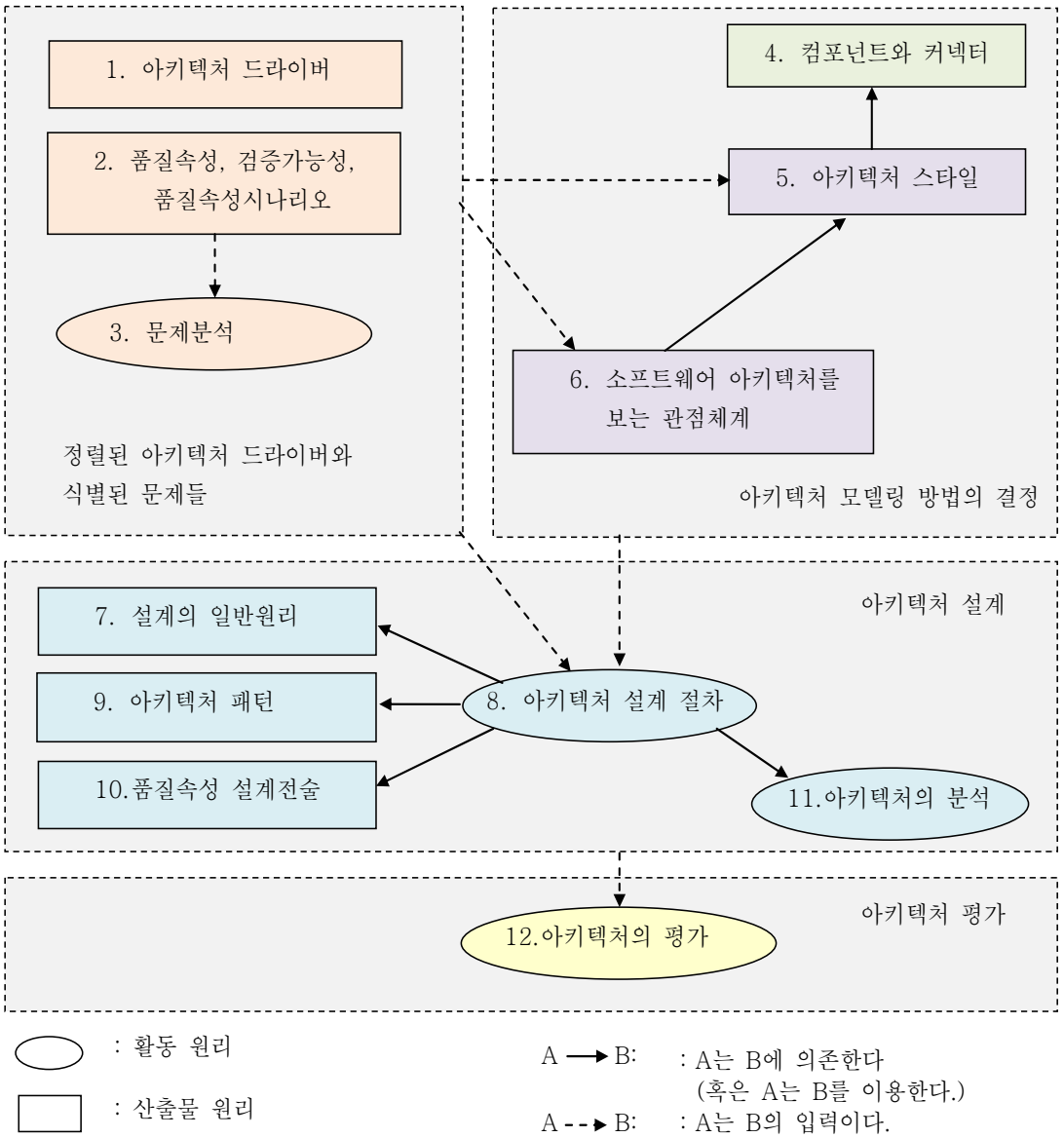

그림 11. 아키텍처 설계절차와 설계에 적용되는 원리[16)]

로써 충족된다. 그러나, 예를 들어, "높은 성능을 가져야 한다"라는 요구사항을 아키텍처 설계에 반영하기가 간단하지 않다. 이러한 요구사항은 일견 문제가 없어 보일 수 있으나, 잠시 생각해 보면 도대체 성능이 무엇을 말하는 것인지, 어느 정도의 성능을 충분히 높은 성능으로 볼 수 있는지 전혀 구체적이지 않다. 이러한 모호성은 당장 아키텍처 설계를 어떻게 해야 되는지에 대한 문제를 남길 뿐 아니라, 최선의 노력을 기울여 개발한 후에도 개발된 시스템에 대하여 고객이 높은 성능을 가진 것으로 판단할지 그렇지 않을 지를 전혀 알 수 없는 문제점을 갖고 있다. 따라서 아키텍처 드라이버들을 먼저 검증할 수 있는 형태로 먼저 바꾸어 놓은 것이 필요하다. 특히 품질속성[17]이라고 불리는 일군의 요구사항들은 아키텍처에 영향을 미치면서, 고객의 도움 없이는 개발자 단독으로 검증 가능한 형태로 바꾸기에 어려운 종류의 요구사항들이다.

검증 가능한 형태의 품질속성은 시나리오의 형태를 갖게 된다. 따라서, 아키텍처 설계에 들어가기에 앞서 품질속성을 검증 가능한 품질속성 시나리오 형태로 바꾸어 주어야 한다.

5.3 문제분석

좋은 아키텍처 설계 절차와 기법들을 우리가 잘 알고 있다고 하여도, 요구사항들이 제기하는 문제를 잘 이해하지 못한다면 올바른 아키텍처 설계가 나올 수 없다. 따라서 주어진 요구사항들이 어떤 문제의 해결을 요청하는 것인지 잘 이해하는 것이 필요하고, 가능한 한 알려진 문제의 형식으로 식별해 낼 수 있다면, 우리가 알고 있는 해법을 적용할 수 있게 됨으로써 문제 해결이 용이해 진다.

5.4 컴포넌트와 커넥터

소프트웨어 아키텍처를 사람들이 이해할 수 있는 형태로 나타내는 것을 소프트웨어 아키텍처 모델(model 혹은 representation)이라고 한다. 소프트웨어 아키텍처는 어떤 모습으로 모델링 되어야 하는 가? 소프트웨어는 궁극적으로 컴퓨팅 자원(data processing, data storage, data transfer)을 활용하여 이용자에게 필요한 서비스를 제공하도록 하는 '컴퓨팅자원의 제어 논리'로 불 수 있다. 그렇다면 컴퓨팅자원의 제어논리인 소프트웨어를 어떻게 나타내어야 하는가? 소프트웨어 아키텍처의 연구자들은 보통 컴포넌트와 커넥터로 표현해야 한다고 말한다. 여기서 컴포넌트는 처리(processing) 혹은 저장(storage) 혹은 전달(transfer)을 수행하는 소프트웨어 시스템의 구성 요소를 말하며, 커넥터는 컴포넌트를 연결하는 연결자(connection element)를 지칭한다.

커다란 컴포넌트의 경우 다시 그 자체가 컴포넌트와 커넥터로 구성될 수 있다. 커넥터는 컴퓨터 메모리의 어느 주소로 이동(jump)하라는 간단한 실행주소변경(transfer)에서부터 시작하여 응용 소프트웨어들을 연결하는 미들웨어에 이르기까지 매우 다양한 종류와 규모가 있다. 한편 미들웨어와 같은 커다란 커넥터는 그 자체가 더 작은 컴포넌트와 커넥터로 이루어진 하나의 컴포넌트이기도 하다. 이와 같이 컴포넌트와 커넥터의 양면성을 가진 가지 대상을 이 논문에서는 특히 커넥션 컴포넌트라고 부르기로 한다.

컴포넌트와 커넥터의 구별은 아키텍처의 표현 형태를 결정짓는 가장 근본적인 시각이며 이는 이어져서 설명되는 아키텍처 스타일과 아키텍처 관점의 적절한 결정에 앞서는 아키텍처 모델링의 기본적인 틀을 제공한다.

---

16) Brooks는 그의 최근 저서[2]의 제 8장 에서 설계의 접근방법을 합리적 방법과 경험적 방법으로 나누었다. 경험과 통찰에 의한 설계가 설계전문가들의 방법이고, Brooks는 설계자들은 "시각적이고 공간적 경향을 가진"오른쪽 뇌가 발달한 사람들이라고 말한다. 이 논문에서 제시하는 설계절차는 합리적 방법과 "시각적이고 공간적인" 방법이 서로 배타적이지 않음을 보여준다.

17) 대표적인 품질 속성으로 성능(performance), 보안성(security), 신뢰성(reliability), 확장성(extendibility), 사용자편의성(usability) 등이 있다.

5.5 아키텍처 스타일

아키텍처 스타일은 아키텍처의 유형을 말한다. 음악에도 재즈, 소울, 락 등의 다양한 “스타일”의 존재하는 것처럼, 아키텍처도 다양한 스타일이 있다는 시각이다. 그리고 적절한 아키텍처 스타일의 선택은 대상시스템의 개발을 효율적이고 효과적으로 만들 수 있다는 시각이다.

같은 스타일의 노래들이 서로 다르면서도 많은 공통점을 가진 것처럼, 같은 아키텍처 스타일을 갖는 시스템들 간에는 많은 공통점이 존재한다. 예를 들어, 파입(pipe)[18]과 필터(filter)라는 두 개의 서로 다른 성격의 컴포넌트로 구성된 다양한 시스템들이 존재할 수 있는데, 이러한 시스템들은 “파입-필터 스타일“을 가졌다고 말할 수 있다.

여기서 주목하여야 하는 것은, 아키텍처 스타일은 그 자체로서 주어진 요구사항을 충족시키는 해법을 제시하지는 않는다는 것이다. 단지 그러한 종류의 요구사항을 충족시키기 위하여는 이런 스타일의 아키텍처가 효과적일 것으로 예견하기 때문에, 그 스타일을 선택하게 되는 것이다. 구체적인 시스템에 도달하기 위하여는 스타일이 주는 장점들을 잘 활용할 필요가 있다. 그러나 시스템의 구체적인 아키텍처를 설계하기 위하여는, 아키텍처 스타일의 결정에 이어 설계의 일반원리나 아키텍처 패턴 혹은 품질속성 설계전술 등의 설계기법 원리들도 적절히 활용하여야 한다. 본 논문에서는 아키텍처 스타일이 직접적으로 설계결과를 주지 않고 간접적으로 효과적인 설계를 지원하기 때문에, 아키텍처 스타일을 설계 단계군에 속하는 설계 기법 원리의 하나로 보지 않고, 아키텍처 모델링 방법의 결정 단계군에 속하는 원리로 본다.

18) 파입은 구성상 필터와 필터 사이에 오기 때문에 이 둘을 연결하는 커넥션 컴포넌트다.

5.6 관점체계

그러면 이러한 입력물로부터 진행되는 설계활동의 궁극적 결과물은 어떻게 표현되어야 (혹은 나타내야) 하는가? 소프트웨어 아키텍처 연구를 통하여 사람들은 소프트웨어 아키텍처는 그 자체로서 우리가 볼 수 있는 혹은 ‘알 수 있는’ 대상이 아니라, 오직 우리가 특정한 ‘관점’으로 볼 때에만, 그 관점의 빛이 투영된 그림자로서 볼 수 있게 된다는 것을 발견하였다. 또한 하나의 관점만 존재하는 것이 아니라 여러 가지 관점이 존재하고 적절한 여러 개의 관점을 동원할 때 아키텍처를 더 잘 포착할 수 있다는 것을 발견하였다. 그렇다면 소프트웨어 아키텍처를 어떠한 관점들로 볼 수 있고, 어떠한 관점들의 집합이 개발 대상 시스템에 대한 적절한 관점들의 집합을 이루게 되는가 하는 문제가 대두된다.

여기서 한가지우리가 주목해야 하는 점은 아키텍처 스타일에 따다 적합한 관점체계가 서로 다를 수 있다는 것이다. 예를 들어, J2EE 기반의 응용, BPMS 기반의 SOA 응용, 내장소프트웨어, Web 응용 등의 아키텍처 스타일에 적절한 관점체계가 모두 동일하지는 않다.

5.7 설계의 일반원리

소프트웨어 아키텍처 설계도 다른 설계와 마찬가지로 적용되는 보편적인 원칙이 있다. 여기서 말하는 설계는, 아키텍처 드라이버부터 아키텍처 평가까지의 아키텍처 설계의 전체 과정을 말하는 것이 아니라, 정렬된 아키텍처 드라이버들로부터 설계선택들의 집합으로 옮아가는 단계를 말한다. 이 단계를 위하여 적용할 수 있는 많은 원리들이 존재하다. 그러나, 그 중에서도 많은 상황에 적용할 수 있고, 대부분의 문제의 해결에 단서가 되는 설계 원리는 무엇인가? 이 방법들은 우리가 의식하건 의식하지 않건 오래 세월을 두고 인간이 생활과

학문에서 생기는 문제들을 해결하는데 적용하여 왔고, 따라서 종종 너무나 당연한 것으로 받아들이기 쉬운 그러한 원리들이다. 이미 제 3절에서 이 들을 그림 11의 설계 절차의 큰 틀을 형성하는 원리로서 소개하였지만, 이 원리는 다시 좁은 의미의 설계를 수행하는 데에도 작용하여야 한다.

5.8 아키텍처 설계절차

아키텍처 설계 절차는, 아키텍처를 결정짓는 요구사항들로부터 먼저 어떤 아키텍처적인 설계의 선택들이 있는가를 판단함으로써 시작된다. 하나의 요구사항에 대하여 여러 가지 설계의 선택이 있을 수도 있고, 여러 개의 요구사항들에 대하여 여러 개의 설계선택들이 존재할 수 있다. 아키텍처 드라이버들을 충족시킬 수 있도록 어떻게 설계선택들을 잘 찾아가고 이들로부터 어떤 판단 하에 구체적인 선택으로 선택의 범위를 압축하여 가느냐 하는 것이 합성과 분석의 기술이다. 다양한 선택들을 찾는 것은 창조적인 과정이고, 그들 중에 장단점을 따져서 좋은 선택으로 좁혀가는 과정이 아키텍처 분석이다.

소프트웨어 아키텍처 설계는 일반적인 원리만이 아니라, 경험적 지식을 활용함으로써 설계의 효율을 높일 수 있다. 이미 검증된 아키텍처 설계를 새로이 발명할 필요는 없다. 경험적 지식은 흔히 많은 사람들에 의하여 오랜 기간 동안 확인된 좋은 해법이다. 그러한 처방(prescription)에는 크게 아키텍처 패턴(혹은 패턴)과 품질속성 공략전술 (혹은 품질속성전술)이 있다.

5.9 아키텍처 패턴

아키텍처 패턴은 반복적으로 발생하는 문제에 대한 미리 만들어진 솔루션으로, 아키텍처 스타일이 구체적인 요구사항의 해결을 제공해 주지 않는데 비하여 패턴은 시스템 전체 혹은 서브시스템에 대한 아키텍처 해법을 제시한다. 따라서, 흔히 패턴에 대한 체계적이고 깊은 이해는 훌륭한 아키텍트의 필수 조건으로 간주되기도 한다.[19]

패턴은 특정한 문제를 해결하는 준비된 처방이며, 품질 속성 설계 전술과 대조적으로 특정 품질속성에 한정되지 않는다. 따라서 아키텍처 설계자들은 설계의 해법을 패턴에서 먼저 찾고, 적절한 해답이 없는 경우 품질속성 설계전술을 사용하게 된다.

5.10 품질속성 설계전술

품질속성 설계 전술(tactic)은 단일 품질속성응답을 제어하는데 영향력 있는 설계결정이다. 시스템이 보통 여러 가지 품질 속성을 동시에 충족시킬 것을 요구하므로, 아키텍처 설계는 보통 여러 개의 요구사항의 충돌을 해결하여야 한다. 패턴은 문제에 대하여 일반적인 솔루션을 제공하여, 여러 가지 요구사항을 동시에 해결할 수 있는 패턴이 존재할 경우 이를 사용하면 되지만, 그렇지 않을 경우, 품질속성 설계 전술을 사용하여 문제를 해결하게 된다. 이 경우 여러 개의 힘의 충돌문제를 추가적으로 해결해야 한다.

5.11 분석

설계의 창조적 단계에 의하여 제시된 주어진 아키텍처가 적절한지 판단하기 위하여는 분석이 필요하다. 예를 들어, 이 아키텍처가 가져다 줄 처리량은 얼마인가와 같은 분석도 가능하다. 분석은 어떤 관점을 가지고 보느냐에 따라, 그에 따른 분석결과가 도출된다. 특별히 제시되는 분석관점을 제외하고 일반적으로 중요한 분석관점으로 위험요인과 민감점(sensitivity point), 상충점(tradeoff point) 등이 있다.

---

19) 물론 이 논문에서는 추가적으로 11가지의 원리를 더 알고 있어야 훌륭한 소프트웨어 아키텍트라고 보는 입장을 취하고 있다.

5.12 평가

아키텍처 분석이 아키텍처에 대하여 주어진 분석관점에서 분석결과를 도출하는 활동인데 반하여, 아키텍처 평가는 아키텍처 분석 또는 다른 방법을 통하여 얻어진 데이터를 종합하여 아키텍처에 대한 중요한 판단을 내리게 하는 활동이라는 점에서 서로 다르다.

문제의 해결방법에는 여러 가지가 존재할 수 있고, 각각의 해법마다 그 장단점이 있기 때문에 아키텍처 평가가 필요할 수 있다. 주어진 아키텍처에 대한 분석은 설계활동 안에서 더 나은 대안들을 찾아가는 과정이며, 평가는 완결된 설계를 대안과 비교함으로써 그 결과에 따라 설계라는 큰 과정을 다시 수행할 것을 요구할 수도 있는 활동이다.

6. 결론

본 논문에서는 효과적인 아키텍처 설계를 위하여 아키텍처 설계 절차를 네 개의 단계군으로 구성된 것으로 보고 아키텍처 설계에 필요한 12개의 근본적인 원리들을 식별하여 개략적으로 설명하였다. 각각의 원리에 대한 보다 상세한 설명과 본 논문에서 제시된 절차와 원리들이 적용되어 효과적으로 사용되는 사례는 다른 기회에 소개할 계획이다.

참고문헌

[1] Bass, L., Clements, P., Kazman, R., Software *Architecture in Practice, 2nd ed.,* Addison-Wesley, 2003.

[2] Frederick R. Brooks, Jr., *The Design of Design, Addison*-Wesley, 2010.

[3] Garlan, D., Perry, D., “Introduction to the Special Issue on Software Architecture,” IEEE Transactions on Software Eng., April 1995.

[4] Institute of Electrical and Electronics Engineers, *Recommended Practice for Architectural Description of Software-Intensive Systems (IEEE Std 1471-2000)*, New York, NY: Institute of Electrical and Electronics Engineers, 2000.

[5] Jacobson, I., Booch, G., Rumbaugh, J., *The Unified Software Development Process*, Addison-Wesley, 1999.

[6] Kruchten, P., *The Rational Unified Process : AnIntroduction, 2nd Ed.*, Boston, MA: Addison-Wesley, 2001.

[7] Kruchten, P., Obbink, H., Stafford, J., “The Past, Present, and Future for Software Architecture,” IEEE Software, Volume 23, Issue 2, March 2006.

[8] Shaw, M., “Larger Scale Systems Require Higher-Level Abstractions,” Proc. Fifth Int’l Workshop on Software Specification and Design, pp 143 ~ 146, IEEE Computer Society, ACM SIGSOFT Software Engineering Notes, Vol. 14, No. 3, May 1989.

[9] Mark Weiser, “The Computer for the Twenty-First Century,” Scientific American, September 1991

저 자 소 개

강 성 원

1982년 서울대학교 사회과학대학(정치학사)
1989년 Univ. of Iowa 전산학(전산학석사)
1992년 Univ. of Iowa 전산학(전산학박사)
1993년~2001년 한국통신 연구개발본부 선임연구원
1995년~1996년 미국 국립표준기술연구소 (NIST)
객원연구원
2001년~2005년 한국정보통신대학교 조교수
2003년~현재 미국 Carnegie-Mellon University
소프트웨어공학석사과정 겸임교수
2005년~2009년 한국정보통신대학교 부교수
2009년~현재 KAIST 부교수

〈관심분야〉 소프트웨어 아키텍처, 소프트웨어 프로덕트 라인, 엔터프라이즈 아키텍처, 소프트웨어 시험, 형식기법

단위 테스트 자동화를 위한 자바 프로그램 테스트 코드 구축

(Building Test Codes for Unit Test Automation of Java Programs)

윤 회 진‡

(Hoijin Yoon)

요 약 애자일 개발의 XP와 Scrum을 중심으로 단위 테스트 자동화의 중요성이 커지고 있다. 그러나 테스트 결과, 즉 통과 또는 실패를 자동으로 결정하기 위해서는 테스트 실행 결과와 예상 결과를 비교하는 과정이 필요하다. 이 부분의 구현이 자동화의 성패를 좌우한다. 본 연구는 단위 테스트 자동화를 위한 테스트 코드 작성을 소개하고, 테스트 코드 구현에서 고려해야할 사항을 언급한다. 첫째, void 형태의 메소드의 경우 테스트 데이터 실행 결과를 명시적으로 구하기 어려운 문제를 본 연구에서는 Mock 프레임워크를 사용하여 해결하였다. 둘째, void 형태의 메소드의 경우, criteria로 인해 구성된 테스트 경로상의 모든 문장들이 제대로 수행되었는지 하나씩 살펴보아야 하는지, 아니면 최종 문장에 대해서만 보아야 하는지의 문제이다. 본 연구에서는 Mock 프레임워크의 verify 기능을 활용하여 매 순간 제대로 실행되어져야 하는 메소드 호출을 중심으로 명확한 매개변수들을 사용하여 호출이 일어났는지를 확인하고, 그 결과들이 모두 예상 결과와 맞을 때, 해당 테스트 케이스에 대한 테스트를 통과한 것으로 결정하였다.

키워드 단위 테스트, 테스트 스텁, Mock 프레임워크, 테스트 자동화, 테스트 주도 개발

Abstract Agile development is mentioned a lot by developers these days. XP or Scrum is one of the popular development processes, and it says that unit test automation would drive an agile development successful. The success of unit test automation depends on how well to compare an execution result to its own expected result. that is why this paper focuses on the comparison part. This paper introduces how to build test codes for unit testing, and then concludes with mentioning two considerations of unit testing automation. First, test codes for void-typed methods need Mock Framework to monitor their behavior. Second, the comparison of execution results and expected results is hard to implement in case of testing void-typed methods. We check every sentences of a test path to decide if the testing is fail or pass.

Key words Unit testing, Test stub, Mock Framework, Test Automation, Test Driven Development

1. 서 론

포레스터 리서치의 2008년 2월 보고서 (Enterprise Agile Adoption in 2007)[1] 에 따르면, 북미 및 유럽 지역에서 약 1/4 가량의 기업들이 애자일을 이미 도입했다. 그리고 애자일 도입의 속도가 가속화되고 있다. (“Enterprise Agile adoption has accelerated, increasing approximately two and a half times faster between 2006 and 2007 than between 2005 and 2006.”) 이 보고서는 또한 기업의 규모(직원수)가 클수록 애자일 더 많이 하는 경향이 있었다고 분석하였다. 예컨대 2만명

‡ 정 회 원 : 협성대학교 컴퓨터공학과 조교수
hjyoon@uhs.ac.kr
논문접수 : 2010년 11월 2일
심사완료 : 2010년 12월 20일

이상의 직원을 갖은 기업의 경우 34%가 애자일을 도입하고 있다.

애자일은 프로젝트의 구체적인 개발 방법론이 아니다. 기존의 전통적인 개발 방법론이 가진 한계를 극복하고자 새롭게 나온 개발 방법론을 통칭하여 부르는 개발 프로세스의 이름이다. 애자일 프로세스에는 여러 방법론이 있는데, 각자의 특징이 있다. Scum 같은 경우는 프로젝트 관리 방법론에 치중하고 있고, XP의 경우는 개발 기술에 치중하고 있다. 애자일 컨설팅 업체 VersionOne이 2009년 7월부터 12월까지 88개국 총2570명을 대상으로 "어떤 애자일 기법을 적용하고 있습니까"를 조사 결과는 다음과 같다 [2]. 이 조사에 언급된 애자일 기법들은 Scrum과 XP에서 제안하는 기법들이다.

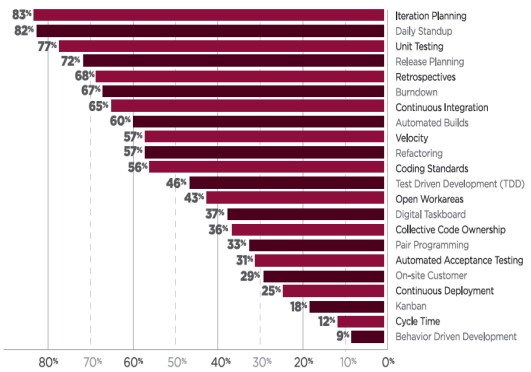

그림 1. 애자일 기법의 사용 분포도 [2]

이 가운데 관리 기법을 제외한 XP중심의 개발 테크닉으로만 순위를 뽑아보면 다음과 같다.

1위: 단위 테스트 (Unit testing)
2위: 지속적 통합 (Continuous Integration)
3위: 자동화된 빌드 (Automated Builds)
4위: 테스트 주도 개발 (TDD)

애자일 개발에서의 "단위 테스트"는 개발자 입장에서 반드시 수행해야 한다. 모든 개발 단위들은 그에 대한 테스트 코드를 함께 가지고 있어야 하며, 이 테스트 코드는 요구사항 변화에 따른 추가 테스트에 사용되어 회귀 테스트 비용 절감에 기여할 수 있다. 또한 요구사항 변화에 민첩하게 대응하는 원동력이 된다. 위의 4위에 해당하는 TDD에서는 특히 테스트 코드를 우선 개발하여야 한다. 이는 테스트를 실행하기 위한 드라이버와 스텁이 모두 갖추어진 테스트 코드를 의미한다. 따라서 테스트 스텁 구현이 필요하다.

본 연구에서는 애자일 등의 최근 개발 방법들의 기법들이 기반으로 하는 단위 테스트 자동화를 위한 테스트 스텁을 구현하고, 이때 실제 개발 현장에서 많이 사용되는 Mock 프레임 워크와 JUnit을 활용한다. 2장에서는 테스트 스텁과 Mock 프레임 워크에 대하여 살펴보고, 3장에서는 실제 JMemorize 프로그램에 대한 테스트 코드를 구현한다. 4장에서는 테스트 자동화를 위한 코드 작성에서 고려해야 할 부분에 대하여 분석하고 5장에서 본 연구에 대한 결론을 언급한다.

2. 관련연구

2.1 테스트 드라이버와 테스트 스텁 (Test Stub)

테스트 드라이버는 테스트할 모듈을 제어하고 동작시키는데 사용된다. 드라이버의 가장 단순한 예들 중 하나는 순차적으로 실행되는 프로그램이나 명령들의 목록인 배치파일이다. 과거의 테스터들은 테스트 모듈의 이름을 포함하는 배치 파일을 만들고 배치 실행을 시작한 다음, 퇴근했다. 오늘날은 xUnit 프레임워크의 등장으로 테스트 드라이버 작성에 큰 도움이 되고 있다. xUnit 프레임워크는 표 1과 같이 다양한 프로그래밍 환경을 도메인으로 맞춤되어져 개발되어져 있다.

표 1. xUnit 프레임워크 기반의 테스트 도구들 [3]

xUnit	프로그래밍언어	관련사이트
CUnit	C	http://cunit.sourceforge.net
CppUnit	C++	http://sourceforge.net/projects/cppunit
PHPUnit	PHP	http://www.phpunit.net
PyUnit	Python	http://pyunit.sourceforge.net
DbUnit	DataBase	http://dbunit.sourceforge.net
utPLSQL	PL/SQL	http://utplsql.sourceforge.net
...	...	...

테스트 드라이버 개념이 xUnit 프레임워크에 포함되어져 있다. xUnit은 테스트 모듈들을 일련의 규칙에 따라 실행하는 코드를 자동 생성시켜줌으로써, 테스트 드라이버 코드 개발을 자동화하고 있다. 테스트 드라이버와 반대 개념의 테스트 스텁은 개발자가 테스트 코드로 개발해주어야 한다. 특히 TDD의 경우 테스트 코드를 미리 작성해야 하므로 테스트 스텁 구현이 필수적이다. 특히 아직 다른 모듈과의 통합이 이루어지지 않은 시기의 단위 테스트에서는 테스트 스텁 코드 구현을 위해 더미(dummy) 모듈을 구현하여 테스트 스텁으로 사용하기도 한다.

Gerard Meszaros의 xUnit Test Pattern[4]에는 "Test Double"이라는 것을 설명하고 있다. 이는 테스트를 수행하기 위해서 실제 모듈 역할을 대체하는 기능을 가진 객체나 컴포넌트를 말한다. 대역의 의미를 갖는 Double이라는 용어를 사용한다. 그림 2의 Test double의 분류에서, Mock Object는 Dummy, Stub, Spy등을 통합해 놓은 것으로 볼 수 있다. Mock은 테스트 스텁의 기능에 검증(Assertion)기능을 추가한 형태로서, 테스트 실행 입장에서 더 의미가 있다.

본 논문에서는 Mock Object를 활용하여 테스트 스텁을 구성하고 동시에 검증하는 환경을 구현하여, 테스트 드라이버 기능을 제공하는 JUnit에서의 활용 방법을 보인다.

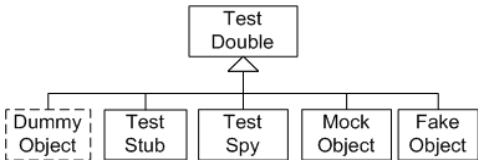

그림 2. Test Double의 분류[4]

2.2 Mock 프레임워크

테스트 더블의 기능을 구현하기 위한 도구로 Mock 프레임워크를 사용할 수 있다. 자바 기반의 Mock 중에서 많이 사용되고, 업데이트가 잘 이루어지고 있는 것에는 EasyMock, JMock, Mockito가 있다. 세계적인 랭킹을 보면, EasyMock과 jMock이 1,2등을 다툰다. 특히 EasyMock의 경우 만들어진지가 오래됐고, 많은 프레임워크에서 EasyMock을 사용했었다. 그리고 Mockito는 역사는 오래되지 않았지만 간편함으로 인해 다크호스로 떠오르고 있다.

Mokito를 개발한 수제빵 파베르(Szczepan Faber)가 설명하는 Mockito의 차별점은 다음과 같다. 첫째, 테스트 그 자체에 집중한다. 둘째, 테스트 스텁을 만드는 것과 검증을 분리시켰다. 셋째, Mock만드는 방법을 단일화했다. 넷째, 테스트 스텁을 만들기 쉽다. 다섯째, API가 간단하다. 여섯째, 프레임워크가 지원해주지 않으면 안되는 코드를 최대한 배제했다. 마지막으로 일곱째, 실패시에 발생하는 에러추적이 깔끔하다. 쉽게 말하면, Mockito는 EasyMock과 jMock의 단점을 보완하기 위해 나온 Mock 프레임워크라고 생각하면 되겠다. 본 연구에서는 기존 기술의 단점을 보완하여 개발된 Mockito를 이용한다.

3. Mockito를 활용한 테스트 스텁 구현

본 절에서는 Opensource로 공개된 JMemorize 소스코드를 단위 테스트한다. 이때 테스트 드라이버의 기능을 하는 jUnit 4와 테스트 스텁으로 Mockito 1.8.2를 이용한다.

3.1 JMemorize

본 연구에서는 자바 오픈소스 프로그램인 JMemorize를 단위 테스트 하기 위한 자동화 환경을 JUnit과 Mokito를 이용하여 구현한다. JMemorize는 플래쉬 카드 방식의 암기 프로그램으로, 라이트너 학습 암기 방식으로 만들어져 유명하다. 인터페이스는 영어와 일어, 에스페란토 등을 지원하고 있으며, 그림 3과 같이 외워야 하는 내용과 그에 대한 암기 기록을 보여주고 있다. 이는 자바 언어로 작성되어져 있다.

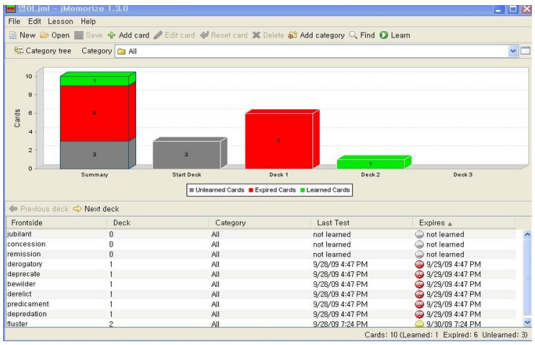

그림 3. JMemorize

이 프로그램은 125개의 클래스로 이루어져 있으며, 각 클래스에 30개 전후의 메소드가 구현되어져 있다. 이에 대한 단위 테스트 환경을 구성한다.

3.2 테스트 코드를 위한 테스트 설계

테스트 코드를 작성하려면 테스트 경로와 그를 트리거하는 입력값, 그리고 그에 대한 예상 결과값을 사용해야 한다. 본 연구에서는 향후 테스트 criteria의 효율성을 분석하기 위하여 화이트박스 테스트 criteria가운데 5개의 criteria를 적용하였다. All-Use, Prime Path, Branch coverage, Edge Pair, Prime Path with sidetrip 등이 그것들이다. 이 가운데 Branch Coverage를 제외한 나머지 4종류의 Criteria에 따른 테스트 경로는 Jeff Offutt이 제공하는 웹도구를 사용하여 도출해 내었다[5]. 즉, 하나의 메소드에 대한 테스트 코드를 5가지 존재하며 각각은 해당 criteria를 통하여 정의된 테스트 경로이다.

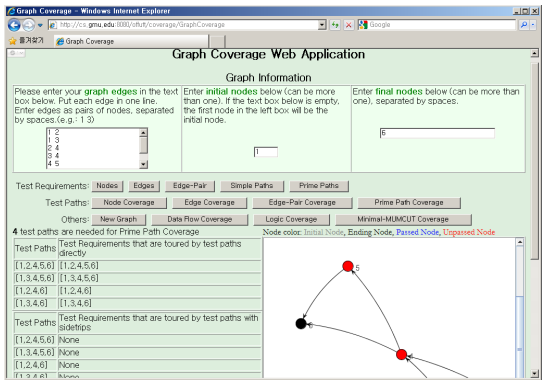

그림 4. Criteria에 따fms 테스트 경로 산출 도구[5]

그림 4에서 보듯이 테스트 대상 메소드의 CFG의 에지들과 시작노드번호, 그리고 종결노드 번호를 입력하고 원하는 criteria를 클릭하면 그에 대한 테스트 경로를 내보내준다. '테스트 코드'는 바로 이 경로를 실행시키는 기능을 한다.

3.3 테스트 코드를 위한 Mokito 활용

Mokito를 Eclipse에 패키지로 추가한후 Eclipse 환경에서 JUnit과 함께 Mokito를 사용한다. Mokito는 앞서 설명한대로, 테스트 스텁을 구현하는 도구이다. 실제 다른 모듈과의 통합 이전의 단위 테스트에 필요한 도구이며, 동시에 테스트 코드를 실행하여 테스트의 자동화를 구현하는데 필수적이다. Mock프레임워크를 어떻게 활용하는지 그림 5가 그 한 예이다.

```
@Test
  public void card_setLearnedAmount_PP_13456()
  { testCard=new Card("Front","Back");
  Category mockedCategory=mock(Category.class);
  testCard.m_category=mockedCategory;
  testCard.setLearnedAmount(false, 100);
  verify(mockedCategory).fireCardEvent(mockedCategor
  y.DECK_EVENT,testCard, testCard.getCategory(),
  testCard.m_level);
  assertEquals(100,testCard.m_backHitsCorrect);
  }
```

그림 5. Mokito 사용 테스트 코드 예

위 테스트 코드는 Card 클래스의 setLearned Amount메소드의 CFG에 Prime Path criteria를 적용한 테스트 경로들 가운데 “1,3,4,5,6”을 실행하는 코드이다. 이 경로를 수행하는 중에 Category 클래스를 호출하여 값을 활용한다. 이 부분을 Mock으로 구현하기 위하여, 다음 코드를 이용한다. 이는 Category 클래스를 흉내내는 Mock 클래스 즉, mockedCategory를 생성하여 테스트 코드에서 이 클래스가 일을 대신하게 하는 것이다.

```
Category
mockedCategory=mock(Category.class);
```

생성된 mock클래스인 mockedCategory는 관찰이 가능해진다. 즉 어떤 테스트 경로상에 mocked Category의한 메소드가 실행되었는지를 확인할 수 있다. 이는 메소드 실행 결과가 특정 값이 아닌 void의 형태로 반환되어 실행되었던 메소드의 실행 여부가 결과가 되는 경우에 매우 유용하다. 그림 5의 테스트 대상 메소드인 setLearnedAmount도 LearnedAmount라는 변수에 특정 값을 설정하는 일을 하며, 그에 대한 어떤 반환값도 없어서, 제대로 수행되는지를 판단하려면 테스트 경로에 따라 잘 진행되었는지를 확인할 필요가 있다. 이때 Mock 프레임워크가 제공하는 verify 메소드를 활용한다. 그림 5에서는 다음과 같이 활용하였다.

```
“verify(mockedCategory).fireCardEvent(mocked
Category.DECK_EVENT,testCard, testCard.getCategory(),
testCard.m_level);”
```

이는 앞서 만든 Mock 클래스인 mocked Category를 사용하여, 테스트 경로상의 메소드인 Category.fireCardEvent 주어진 매개변수들로 실행되었는지를 확인해준다. 이처럼 Mock 프레임워크를 활용하면, 단위 테스트 대상인 메소드를 다른 메소드와의 실제 연결없이 해당 메소드만을 테스트할 수 있다. 이처럼 Mock 클래스를 생성하여 연결하고자 하는 다른 클래스 역할을 대신하게 하고, 또한 그의 행위 전체가 관찰될 수 있게 하여 테스트 경로 상의 작업들이 정확히 잘 수행되었는지를 확인할 수 있는 부가효과가 있다.

3.4 테스트 코드 실행

그림 6과 같이 테스트 코드는 각 criteria별로 패키지를 달리하고 있게 구현하였다. src패키지에는 JMemorize 소스코드들이, Branch_test에는 Branch coverage에 따라 구성된 테스트 경로가 실행될 수 있는 테스트 코드가 테스트 대상 클래스 단위로 하나의 파일로 구성되어져 있다. 예를 들어 Branch_CardTest.java파일은 Card 클래스의 메소드들을 위한 테스트 코드들이다. EdgePair_test, AllUse_test, SideTrip_test, PP_test는 각각 Edge Pair, All Use, Prime Path with SideTrip, 그리고 Prime Path Criteria를 구현하고 있다.

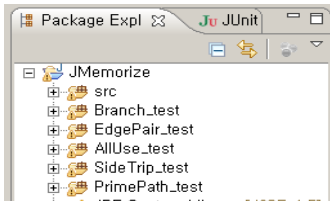

그림 6. Criteria 단위의 패키지

이 가운데 Prime Path를 적용한 테스트 코드를 실행시키보면 그림7의 결과로 보인다. Card 클래스에 속한 메소드들의 테스트 경로가 JUnit의 Run 버튼 하나로 실행되며, 그 결과 하나의 실패, 즉 예상출력과 다른 결과가 나온 테스트 케이스가 발생했음을 보이고 있다.

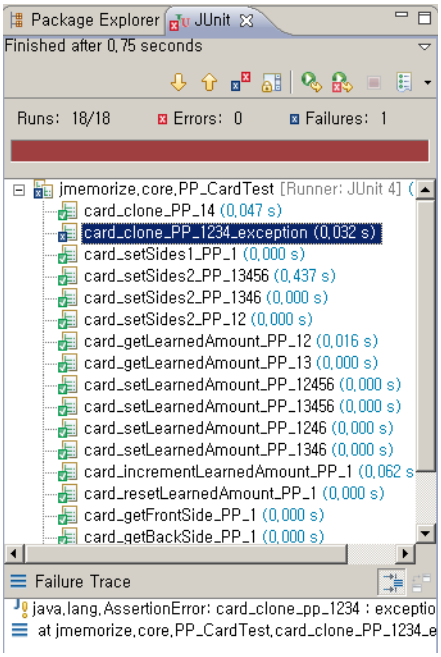

그림 7. 테스트 코드 실행 예

4. 단위 테스트 자동화

본 연구에서는 단위 테스트 자동화 환경을 구축하였다. 자동화를 하기 위하여 테스트 코들 구현해야 하며, 그 코드는 실행 가능해야 한다. 이때 테스트 드라이버와 테스트 스텁이 필요해지는데, 본 연구에서는 JUnit을 테스트 드라이버로, Mock 프레임워크를 테스트 스텁으로 활용하였다.

4.1 "테스트 주도 개발"에서의 단위 테스트

단위 테스트 자동화는 앞서 살펴본 대로 현재의 소프트웨어 개발 문화에서 요구되는 기술이다. 빠른 릴리즈를 목적으로 하는 애자일 접근법으로 소프트웨어를 개발하고자 하는 노력이 커지고 있으며, XP 와 Scrum등의 개발 방법론을 중심으로 그에 따른 구체적인 개발 기술들이 대두되고 있다. 이 기술들의 공통점은 모두 테스트 자동화를 필요로 하고 있다. 특히 테스트 주도 개발 (Test-Driven Development : TDD)의 경우, 모듈 개발 이전에 그를 위한 테스트 코드를 우선 개발하는 방법으로서, 테스트 코드가 다른 모듈과의 실제 연결 없이 단위 모듈에 충실하게 작성될 필요가 있다. 이때 Mock 프레임워크가 필요하다[6].

그림 8은 TDD의 구조를 설명하고 있다. 먼저 작성한 테스트 스크립트가 'Green'이 되도록 코드 작성 및 수정을 반복하는 과정을 거쳐서 일단 테스트가 'pass' 되도록 한다. 그 이후 리팩토링 과정을 거쳐 코드의 품질을 향상시킨다. 앞서 언급한대로 대부분 위의 3번 과정을 소홀히 하여 TDD로 작성된 코드의 유지보수성을 떨어뜨리는 결과를 갖게 된다. TDD를 제대로 하려면 '테스트'에 대한 지식과 더불어 마지막 단계인 '리팩토링'에 대한 기술도 적용해야 한다. 리팩토링은 코드가 원래 하는 일이 변경되지 않으면서 내부적인 코드의 구조를 재구성하는 규율에 따른 기술이다(A disciplined technique for restructuring an existing body of code, altering its internal structure without changing its external behavior.)[7]. 리팩토링에 대하여 아는 개발자가 자신의 코드를 이해하기 쉽고 유지보수성이 높도록 만들어야 한다.

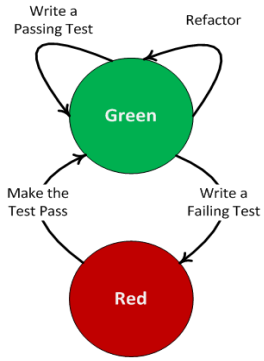

그림 8.TDD 개발 구조

위의 전반적인 과정이 이루어지기 위해서는 해당 단위 모듈마다 테스트 코드 작성이 선행되어져야 한다. 본 연구는 이에 더 나아가 애자일의

다른 개발 기술인 “지속적 통합”과 “자동 빌드” 기술을 지원하기 위하여 단위 테스트 코드 자동화 문제를 고려하였다. 단위 테스트 코드 자동화는 테스트 코드 그 자체가 단독 실행가능하며, 그 결과가 테스트의 통과 및 실패로 명시되어야 한다.

4.2 단위 테스트 자동화의 제약점

본 연구에서 시도했던 단위 테스트 자동화 실험을 통해, 테스트 자동화의 몇가지 제약점을 도출해 내었다. 그동안 단위 테스트 자동화는 테스트 케이스 내의 예상 출력 부분을 도출하는 문제가 극복되어야할 문제도 언급되었었다. 그러나 그 이외에 실제 자동화 코드 구현을 진행하는 동안 고민했던 몇가지 부분들은 다음과 같다.

첫째, 테스트 실행 결과가 테스트 예상 결과와 같은지를 비교하는 코드 작성이 단순하지 않다. 기존의 JUnit을 이용한 단위 테스트 예제들은 대부분 테스트 대상 메소드의 결과가 int 또는 double 등과 같이 명시적인 반환값을 갖는 경우들이다. 그림 9의 (a)는 string의 반환값을 갖는 getBackSide 메소드의 테스트 코드의 예이고, (b)는 의 경우는 void 메소드인 setLearnedAmount의 테스트 코드이다. (a)의 경우가 일반적인 JUnit 사용법에 등장하는 형태이며, void의 경우는 이 방법을 사용할 수 없다. (b)는 해당 메소드가 테스트 경로를 따라 제대로 실행되고 있는지를 관찰하는 Mock 객체를 생성하여 그 결과를 검증하는 방법을 사용하였다. (a)의 경우보다 (b)의 경우가 구현이 복잡하고, 그림 9의 (a)에서의 getBackSide 형태의 메소드 이상으로 그림9의 (b)에서의 setLearnedAmount의 형태의 메소드가 많이 쓰인다. JMemorize의 Card클래스의 경우, 총 40개의 메소드 가운데, 명시적으로 String 또는 int를 반환하는 메소드는 9개, 다른 클래스의 형태로 반환하는 경우는 10개, 그 나머지 21개의 메소드는 void이다. 즉, 50% 이상의 메소드가 void형태, 즉 명시적으로 반환값이 없는 형태를 가지므로, 반환값이 예상대로 나왔는지를 확인하는 방법으로는 테스트 결과가 실패인지 통과인지 판단할 수 없다. 쉽게 예상되는 장벽이었음에도 단순히 JUnit을 활용하면 단위 테스트가 해결될 거라는 기대감이 있었다. JUnit이 이 문제를 어떻게 처리하고 있는지를 확인하기 위하여 자료들을 찾아보았으나, 그에 대한 참고 자료로서의 문헌은 발견하지 못했으며, 단지 Mock을 사용해야 한다는 것과 행위 기반 테스팅에 대한 정보를 얻었다. void 메소드의 경우처럼 테스트 대상 메소드가 명확한 결과를 내지 않는 경우, 실행 도중의 행위들을 스파이 객체 등을 구현하여 관찰하고 그 관찰 결과가 예상과 같은지를 테스트하는 것이다. 이때 Mock 프레임워크가 사용된다.

```
@Test
public void card_getBackSide_PP_1(){
Card testCard=new Card("Front","Back");
assertEquals(testCard.m_backSide,testCard.getBackSide());
}
```

(a)

```
@Test
public void card_setLearnedAmount_PP_1346(){
testCard=new Card("Front","Back");
Category mockedCategory=mock(Category.class);
testCard.m_category=null;
testCard.setLearnedAmount(false, 100);
verify(mockedCategory,never()).fireCardEvent(mockedCateg
ory.DECK_EVENT, testCard, testCard.getCategory(),
testCard.m_level);
assertEquals(100,testCard.m_backHitsCorrect);
}
```

(b)

그림 9. 테스트 코드의 두가지 경우

둘째, 어디까지 테스트 데이터의 실행 결과로 볼 것인가의 문제이다. 명확한 반환값을 갖지 않는, 앞서 언급한 void 형태의 메소드의 경우, criteria로 인해 구성된 테스트 경로상의 모든 문장들이 제대로 수행되었는지 하나씩 살펴보아야 하는지, 아니면 최종 문장에 대해서만 보아야 하는지의 문제이다. 그림 9의 (a)는 명확한 반환값이 명시되어져 있는 메소드를 테스트하는 경우로서 그 실행 결과도

assertion을 사용하여 반환값이 제대로 나오는지를 검증하여 테스트 결과를 결정한다. 그에 반해 그림 9의 (b)는 Category클래스의 fireCardEvent메소드가 주어진 매개변수들을 가지고 실행된 적이 없는지를 확인한 후 또 Card 클래스의 m_backHitsCorrect의 값이 100과 같은지를 assertion으로 확인한다. (b)의 경우는 두가지 내용에 대하여 확인하여 두 경우 모두 통과된 경우, 메소드가 테스트를 통과하였다고 결정짓는다. 여기에서 (b)의 코드가 충분히 테스트 결과를 확인하는지에 대한 의구심이 들 수 있다. 실제 테스트 코드를 작성하면서 테스트 결과를 테스트 코드에서 어느 정도 구체적으로 또는 자세히 확인해야 테스트 결과를 결정하는데 의미가 있을까에 대한 판단 기준이 필요하였다. 이에 대한 보다 객관적이고 검증된 자료가 요구됨을 알고, 향후 이 문제에 대한 연구를 진행하기로 한다.

5. 결론

애자일 개발에 대한 관심이 커지면서, XP와 Scrum을 중심으로 하는 개발 기술들이 대두되고 있다. 이들의 대부분은 단위 테스트를 중요하게 고려하고 있으며, 덧붙여 자동화를 필수 요소로 내세우고 있다. 이에 따라 단위 테스트 자동화에 대한 논의가 일고 있으나, 지금까지 테스트 자동화는 테스트 데이터를 적용하는 도구 수준에 머물고 있었다. 그러나 실제 테스트 결과, 즉 통과 또는 실패를 결정하기 위해서는 테스트 실행 결과와 예상 결과를 비교하는 과정이 필요하다. 이 부분의 구현이 자동화의 성패를 좌우한다.

본 연구는 단위 테스트 자동화를 위한 테스트 코드 작성을 과정을 진행하는 과정을 소개하고, 테스트 코드 구현에서 해결해야 하는 문제들을 언급하고 있다. 단위 테스트 도구로서의 JUnit을 사용하면서 반환값이 명시적이지 않은 void형태의 메소드를 테스트 할 때, 실제 테스트 현장에서 만나게 되는 문제로 다음의 두 가지를 도출해 내고, 그에 대한 현재의 해결 방안을 설명했다. 첫째, void 형태의 메소드의 경우 테스트 데이터 실행 결과를 명시적으로 구하기 어려운 문제를 본 연구에서는 Mock 프레임워크를 사용하여 해결하였다. 둘째, void 형태의 메소드의 경우, criteria로 인해 구성된 테스트 경로상의 모든 문장들이 제대로 수행되었는지 하나씩 살펴보아야 하는지, 아니면 최종 문장에 대해서만 보아야 하는지의 문제이다. 이 문제는 본 연구에서는 Mock 프레임워크의 verify 기능을 활용하여 매 순간 제대로 실행되어져야 하는 메소드 호출을 중심으로, 명확한 매개변수들을 사용하여 호출이 일어났는지를 확인하고, 그 결과들이 모두 예상 결과와 맞을 때, 해당 테스트 케이스에 대한 테스트를 통과한 것으로 결정하였다.

결국 단위 테스트 자동화의 문제는 테스트 대상 모듈의 테스트 데이터 적용 이후의 결과를 비교하는데 있다. 명시적인 반환값이 있는 경우는 어렵지 않게 해결할 수 있고 모두 그 경우에 대한 내용으로 단위 테스트를 설명하고 있다. 그러나 실제 명시적 반환값이 없는 테스트 void 형태의 메소드가 많은 부분을 차지하고 있다. 실험에 사용된 JMemorize의 하나의 클래스의 경우, 총 40개의 메소드 가운데, 명시적으로 String 또는 int를 반환하는 메소드는 9개, 다른 클래스의 형태로 반환하는 경우는 10개, 그 나머지 21개의 메소드는 void이다. 즉, 50%이상의 메소드가 void형태이며, 다른 클래스를 반환하는 10개의 메소드들도 하나의 값으로 그 결과를 판단하기 어려운 상황이다.

향후 명시적 반환값을 갖는 메소드와 아닌경우의 분포를 많은 수의 클래스들을 분석하여 통계로 향후 구축할 계획이며, 테스트 결과 분석 대상을 어느 수준으로 해야 하는지에 대한 판단 기준에 대한 연구도 진행하고 있다.

참 고 문 헌

[1] 포레스터 리서치, Enterprise Agile Adoption in 2007, 2008

[2] VersionOne, 3rd Annual Survey: State of Agile Development Survey, http://pm.versionone.com/whitepaper_AgileSurvey2008.html, 2009

[3] 이상민, 자바 개발자도 쉽고 즐겁게 배우는 테스팅 이야기, 한빛미디어, 2009

[4] Gerard Meszaros, Xunit Test Patterns: Refactoring Test Code, Addison-Wesley Professional, 2007

[5] Jeff Offutt, Graph Coverage, http://cs.gmu.edu:8080/offutt/coverage/GraphCoverage

[6] 채수원, 테스트 주도 개발 TDD 실천법과 도구, 한빛미디어, 2010

[7] Martin Fowler, Refactoring : Improving the Design of Existing Code (Improving the Design of Existing Code), Addison-Wesley Professional, 1999

저자소개

윤 회 진

1993년 이화여자대학교 전자계산학과(학사)
1998년 이화여자대학교 컴퓨터학과(석사)
2004년 이화여자대학교 컴퓨터학과(박사)
2004년~2005년 Georgia Institute of Technology (Post Doc.)
2005년~2007년 이화여자대학교 컴퓨터학과 전임강사
2007년~현재 협성대학교 컴퓨터공학과 조교수

〈관심분야〉 소프트웨어 테스팅, 요구사항 검증, 애자일 테스팅

컴포넌트 그리드: 개발자 친화적인 국방 소프트웨어 재사용 지원 환경†

(Component Grid: A Developer-centric Environment for Defense Software Reuse)

고 인 영¶ 구 형 민‡
(In-Young Ko) (Hyung-Min Koo)

요 약 국방 소프트웨어 개발 분야는 응용 도메인이 다양하며 각 도메인의 규모도 다른 분야보다 커서 소프트웨어 자산의 재사용이 중요시 되고 있고, 재사용 되는 자산의 품질 및 신뢰성이 강조된다. 국방 분야에서 이러한 중요성을 인식하고 재사용 방법론들을 개발하여 사용하려는 시도가 많았지만 체계적인 재사용이 이루어지지 않아 재사용 이익을 극대화 하지 못하였다. 본 연구에서는 실질적으로 재사용이 왜 잘 이루어지지 않는지에 대한 문제점들을 분석하고, 이러한 문제점들을 해결할 수 있는 요구사항들을 정리하였다. 이러한 요구사항을 만족하며 전군적 소프트웨어 재사용을 지원하는 개발자 친화적인 재사용 지원 환경인 컴포넌트 그리드 시스템을 개발하고 있다. 컴포넌트 그리드 시스템 개발을 위해 아키텍처를 설계하였고, 아키텍처를 구성하는 세부적인 핵심 요소들과 그 역할을 정의하였다. 개발자 부담의 감소를 위한 시맨틱 태깅(Semantic Tagging) 기반의 요구사항 추적 기술을 개발하였고, 재사용 지식 표현 모델을 개발하였다. 또한 개발자들 간의 자유로운 의사소통 과 자산 및 지식의 교류를 지원하기 위해 웹 기반의 자산 관리 환경과 소셜 네트워크 기반의 자산 검색 및 커뮤니티 추천 기법, 위키(Wik) 기반의 참여적, 협력적 지식 정제 및 증식 환경을 개발하였다. 이러한 접근법들을 통합하여 재사용을 지원할 수 있는 웹 기반 컴포넌트 그리드 시스템의 프로토타입을 구현하였다. 본 연구를 통해 국방 소프트웨어 개발 분야에서 개발자들이 소프트웨어 자산들을 투명하고 효율적으로 공유 및 재사용이 가능하게 하여 국방 소프트웨어의 재사용성 및 품질을 향상시킬 수 있으리라 기대한다.

키워드 소프트웨어 재사용, 소프트웨어 재사용 지원 환경, 국방 소프트웨어

Abstract In the defense software domain where large-scale software products in various application areas need to be built, reusing software is regarded as one of the important practices to build software products efficiently and economically. There have been many efforts to apply various methods to support software reuse in the defense software domain. However, developers in the defense software domain still experience many difficulties and face obstacles in reusing software assets. In this paper, we analyze practical problems of software reuse in the defense software domain, and define core requirements to solve those problems. To meet these requirements, we are currently developing the Component Grid system, a reuse-support system that provides a developer-centric software reuse environment. We have designed an architecture of Component Grid, and defined essential elements of the architecture. We have also developed the core approaches for developing the Component Grid system: a semantic-tagging-based requirement tracing method, a reuse-knowledge representation model, a social-network-based asset search method, a web-based asset management environment, and a wiki-based collaborative and participative knowledge construction and refinement method. We expect that the Component Grid system will contribute to increase the reusability of software assets in the defense software domain by providing the environment that supports transparent and efficient sharing and reuse of software assets.

Key words Software reuse, Software reuse-support environment, Defense software

1. 서 론

국방 소프트웨어 분야는 응용 도메인이 다양하며 각 도메인의 규모도 다른 분야보다 일반적으로 크다. 그래서 재사용의 가능성이 타 분야보다 높으며 재사용되는 컴포넌트 품질의 중요성도 강조된다. 그동안 국내에서 컴포넌트 기반 소프트웨어 개발 방법론이 널리 적용되어 사용되어 왔고, 국방 분야에서도 이러한 방법론들을 기반으로 많은 컴포넌트들이 제작되어 왔다. 그러나 이러한 컴포넌트들의 체계적인 재사용이 이루어지지 않아 재사용에 대한 이익을 극대화 하지 못하였다. 또한 서로 다른 시스템 통합 업체들이 국방 소프트웨어 개발에 참여하게 되어 자사의 개발 방법론 및 개발 도구들을 활용한 소프트웨어 개발이 이루어져 재사용을 더욱 어렵게 하였다. 그리하여 국방 소프트웨어 개발비용은 계속 증가하게 되었고 최근 컴포넌트 재사용의 중요성을 다시 파악하게 되어 체계적인 재사용 절차 정의 및 재사용 지원 환경 구축의 필요성을 인식하게 되었다.

국방 도메인에서 소프트웨어 재사용이 왜 어려운지 문제점을 보다 구체적으로 파악하기 위해 육군본부, 공군본부, 국군지휘통신사령부 등의 실 개발부대를 방문하여 설문과 인터뷰를 진행하였다. 설문 및 인터뷰 내용은 현재 소프트웨어의 재사용 현황을 분석하고 향후 재사용 수요를 조사하며 실질적인 재사용 관련 문제가 무엇인지에 대한 것이었다. 설문 및 인터뷰를 통해 상호운용성 포털 시스템, 육군소프트웨어 관리 시스템 등의 소프트웨어 관리 도구는 존재하고 있고 운용 및 관리 측면에서는 잘 활용되고 있지만 등록된 자산들이 효율적으로 재사용 되고 있지는 않다는 것을 알 수 있었다. 이는 국방 도메인 특성상 개발에 다양한 스테이크홀더(stakeholder)들이 참여하고 있고, 분산된 개발 조직 구조 및 절차를 가지며 물리적 혹은 기능적으로 분권화되어 있는 개발 그룹이 그 원인으로 분석되었다.

이러한 환경으로 인해 소프트웨어 개발자들은 재사용 가능한 자산을 어디서 찾아할지 모르는 어려움에 직면하게 된다. 현재 이용되고 있는 국방 소프트웨어 관리 환경을 분석해 보면 단순한 업로드 및 다운로드 기능, 간략한 자산의 명세만 지원하고 있어 재사용할 자산을 찾는다 하더라도 그 자산이 자신의 목적에 맞는지의 여부를 판단하기 힘들다. 그리고 의사결정 이후 실제 자산을 재사용함에 있어서 자산의 개발 담당자나 재사용 관련 정보를 파악하기 쉽지 않기 때문에 소프트웨어 자산을 올바르게 재사용하기 어려운 문제점 또한 발생한다. 이러한 문제점들을 해결하기 위해서는 전군적으로 소프트웨어 자산을 재사용 할 수 있도록 지원해 주는 환경의 제공이 필요하다. 이러한 환경을 지원하기 위한 요구사항을 아래와 같이 분석하였다.

- **효율적인 의사소통 방법 제공** : 국방 소프트웨어 개발 환경은 분산된 환경과 분권화된 개발 조직을 갖기 때문에 개발자들 간의 의사소통과 정보 공유, 피드백 교환 등이 어렵다. 장소에 구애 받지 않고 개발자들 간에 자유롭게 의견 및 정보를 주고받을 수 있는 방법이 제공되어야 한다.
- **소프트웨어 자산의 상호 교류 및 공유 지원** : 현재 분권화 되어 개발이 이루어지고 있기 때문에 조직간 서로 재사용 가능한 자산이 있는지 여부를 알기 어렵고 중복된 개발이 이루어지는 경우도 있다. 재사용 가능한 자산을 조직간, 개발자들 간에 서로 교류하고 공유하도록 지원해 주는 방법이 필요하다.

† 본 연구는 방위사업청과 국방과학연구소의 지원으로 수행되었습니다. (2010-SW-12-DM-01)
¶ 종신회원 : 한국과학기술원 전산학과 부교수
iko@kaist.ac.kr
‡ 학생회원 : 한국과학기술원 전산학과
hmkoo@kaist.ac.kr
논문접수 : 2010년 11월 16일
심사완료 : 2010년 12월 22일

- **자산 재사용에 관한 가이드라인 제공** : 재사용할 자산을 찾았다 하더라도 어떻게 올바르게 재사용해야 하는지 충분한 정보가 없으면 개발자들은 어려움에 직면하게 된다. 이러한 어려움의 해결을 위해 자산 재사용 방법에 관한 적절한 정보를 제공할 수 있는 방법이 필요하다.

- **개발자의 부담을 감소하여 주는 재사용 도구의 지원** : 개발자들은 소프트웨어 자산을 재사용 관리하고 활용하는 것을 부가적인 작업으로 여기는 경향이 있어 부담을 가지게 된다. 개발자들이 재사용 활동을 함에 있어서 부담을 최소화하고 쉽게 사용할 수 있는 도구의 제공이 필요하다.

본 연구에서는 이러한 요구사항들을 만족시킬 수 있는 재사용 환경을 제공하기 위하여 '웹 기반 국방 소프트웨어 컴포넌트 그리드' 시스템을 개발 중에 있고, 시스템의 실제 적용을 위해 국방과학연구소와 과제를 수행 중에 있다. 연구의 목표는 국방 소프트웨어 도메인에서 개발자들이 소프트웨어 자산들을 투명하고 효율적으로 공유 및 재사용이 가능하게 하는 개발자 친화적인 환경을 제공하는 것이다.

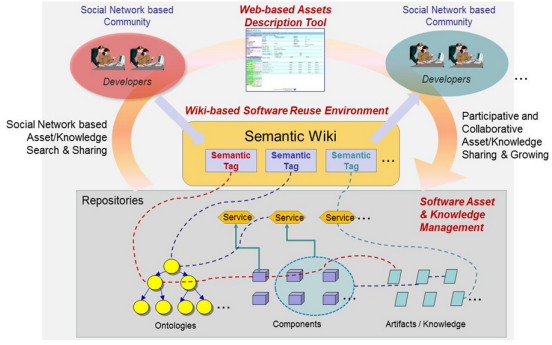

그림 1. 개발자 친화적 재사용 환경 개요[1]

그림 1은 웹 기반의 개발자 친화적인 재사용 환경의 개요를 보여준다. 본 연구에서는 개발자들 간의 참여와 협력을 유도하고 사용하기 쉬운 환경을 제공하기 위해 위키(Wiki) 기반의 재사용 환경을 제공한다[2]. 위키 기반의 인터페이스를 통해 개발자들은 참여적, 협력적으로 자산 및 관련지식을 공유하고 증식 시켜 나갈 수 있다. 자산의 등록 및 명세를 위한 웹기반 재사용 자산 관리 도구 또한 제공한다. 개발자들 간의 자유로운 의사교환을 위해 소셜 네트워크 기반의 커뮤니티 생성 및 관리를 지원하고, 소셜 네트워크 기반의 자산 검색 방법 또한 제공한다. 개발자들의 관련 자산 검색에 대한 부담을 줄여주기 위해서 시맨틱 태깅(Semantic Tagging) 기반의 자산 추적 및 추천 기술을 제공하고, 재사용의 어려움 해결 및 가이드라인을 제공하기 위하여 지식 형태의 정보를 저장, 관리 및 제공할 수 있는 방법을 제시한다.

웹기반 개발자 친화적 환경 개발을 위해 컴포넌트 그리드 시스템의 아키텍처를 정의하였고, 아키텍처의 세부 요소 및 역할을 정의하였다. 자산 추적 및 추천을 위해서 시맨틱 태깅 기반의 요구사항 추적 기술을 개발하였고, 유사성, 친화성, 친밀성 관점에 따른 소셜 네트워크 기반의 자산 검색 방법을 개발하였다. 재사용 지식 및 가이드라인의 제공을 위해 재사용 지식 표현 모델을 개발하였고, 이러한 기술들을 통합하여 재사용을 지원해주는 위키 기반의 재사용 환경 프로토타입을 구현하였다. 본 연구를 통해 국방 소프트웨어 개발자들 간의 자유로운 의사소통을 가능케 하고, 참여적, 협력적으로 자산 및 지식을 공유 및 증식하게 지원하여 줄 수 있다. 개발자 친화적 재사용 환경은 개발자들의 부담은 감소시키고, 소프트웨어 자산의 재사용성은 증대시킴으로써 전군적 소프트웨어 재사용을 위한 환경을 제공할 수 있으리라 예상한다.

본 논문의 구조는 다음과 같다. 다음 절에서는 관련 연구들로서 웹 기반 소프트웨어 재사용 지원 도구와 소프트웨어 개발자들을 위한 소셜 네트워크 지원 도구들에 관해 설명하고, 3절에서는 컴포넌트 그리드 시스템을 위해 설계된 아키텍처와

구성요소들에 대해 설명한다. 시맨틱 태깅 기반의 요구사항 추적 기술을 4절에서 설명하고, 5절에서는 소셜 네트워크 기반의 재사용 자산 검색 기법에 대해 논한다. 6절에서는 소프트웨어 재사용 지식 표현을 위한 모델에 대해 설명하고 7절에서 구현된 컴포넌트 그리드의 프로토타입을 설명한 후, 마지막으로 결론 및 향우연구에 대해 논한다.

2. 관련연구

본 절에서는 개발자 친화적인 재사용 환경 지원을 위해 웹과 위키를 활용한 연구, 개발자들 간의 협력을 위해 소셜 네트워크를 활용한 관련 연구에 대해 논한다.

2.1 웹기반 소프트웨어 재사용 지원 도구

웹의 편리성, 친화성이라는 장점 때문에 소프트웨어 공학 분야에서 웹 관련 기술들이 많이 활용되고 있으며 소프트웨어 재사용 분야를 위한 웹 기반의 환경 개발을 위한 연구들도 진행되고 있다. 대표적인 사례로 나사(NASA)의 재사용 포털을 들 수 있다[3]. 이는 지구 과학 관련 소프트웨어 재사용 지원 도구로서 개발자들 간 필요한 자원들을 서로 공유하고 보다 쉬운 재사용을 위한 정보를 제공한다. 재사용 포털의 특징은 개발자들의 수준과 목적에 따라 세가지 단계의 패키지화된 재사용 자산을 제공한다는 것이다. SRS(Software Reuse System)은 시맨틱 웹 기술을 기반으로 하는 재사용 지원 도구로서 개발자의 현재 개발 상황에 맞는 재사용 지식과 자산을 추천하여 주는 것을 그 목표로 하고 있다[4]. Revyu는 사용자들이 평론을 원하는 항목에 대해 자유롭게 평론하고 등급을 부여할 수 있는 사이트로서 특히 웹2.0의 매쉬업(Mash-up) 기능을 제공하여 사용자들이 필요한 서비스들을 조합할 수 있도록 지원한다[5].

웹을 이용한 소프트웨어 재사용 환경 중에서 특히 위키를 활용한 관련연구들이 있다. 이는 협력적으로 재사용 관련 자산과 지식을 증식하고 정제해 나가서 소프트웨어의 품질과 신뢰성을 향상시킴으로써 재사용율을 높이고자 하는 것이다[2]. Wikitology는 시맨틱 위키의 개념을 기반으로 자가 구성 재사용을 지원하기 위함이다[6]. 이를 위해 온톨로지 기반의 지식 통신, 취합, 추론 기능을 제공한다. Ontobrowse는 재사용 지식을 검색 및 구조화하기 위해 비정형적, 정형적 문서 관리 방법을 제공하고, 문서들의 기술을 위한 웹 인터페이스를 제공한다[7].

2.2 소프트웨어 개발자 소셜 네트워크 지원 도구

현재 대중들에게 활발하게 이용되고 있는 트위터(Twitter), 페이스북(Facebook) 등의 장점을 활용하여 소프트웨어 개발자들 간의 소셜 네트워크를 형성하게 지원하여 줌으로써 재사용성을 향상시키고자 하는 관련연구들이 진행중에 있다. 데브피아(Devpia)는 개발자들이 등록된 재사용 문제를 해결하면 인센티브(Incentive)를 부여함으로써 재사용 문제 해결 활동에 참여를 유도한다[8]. 데브멘토(Devmento)는 개발자(Developer)와 멘토(Mento)를 합친 용어로 개발자의 수준에 따라 등급을 부여하고 각 등급에 맞는 멘토를 추천하여 줌으로써 재사용에 관한 문제 해결 및 지식을 배울 수 있도록 지원하여 준다[9]. 고수(Gosu)는 아티클(Article) 기반으로 고급자들이 아티클을 작성할 수 있도록 지원해 주는 커뮤니티이다[10]. 고수에서는 카테고리의 분류에 따른 템플릿(Template)을 제공하여 고급자들이 아티클을 작성하여 저장하면 초급자나 중급자가 이를 활용하여 도움을 얻을 수 있다. 스택 오버플로우(Stackoverflow)는 프로그래밍에 대한 전반적인 질문과 응답을 신속할 수 있게 지원해 주는 환경이다[11]. 위키 기반의 협업 환경도 지원

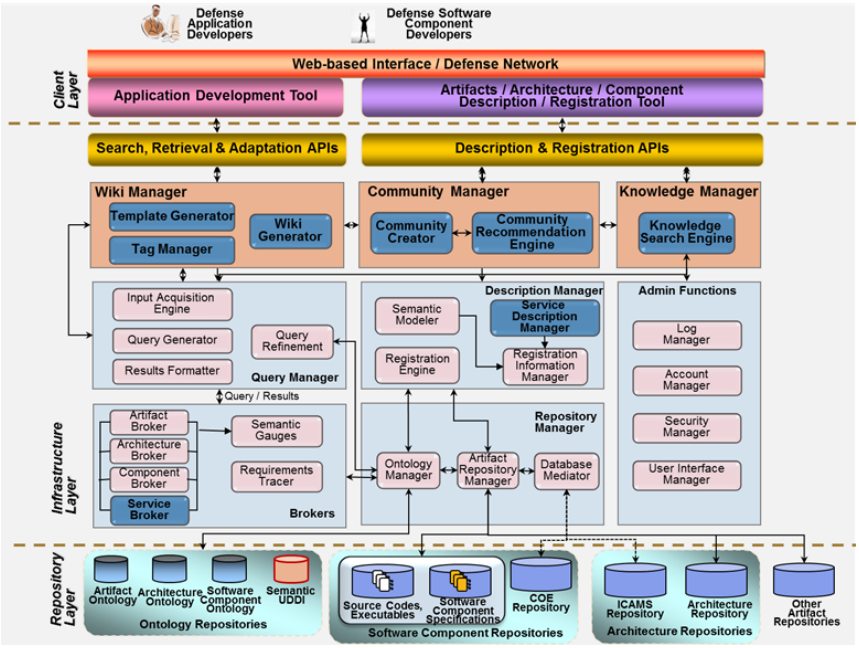

그림 2. 컴포넌트 그리드 아키텍처

하고 개인적 정보 유지를 위한 블로그, 토론을 위한 포럼을 제공한다.

현재 웹이나 위키 기반의 소프트웨어 재사용 지원도구들은 웹 기술을 적절히 활용하여 재사용 환경은 지원해 주고 있기는 하나, 웹 환경의 적용 수준이고 사용자의 부담을 감소하거나 적절한 자산의 추천, 재사용 가이드라인 제공 등에 있어 한계점이 있다. 소셜 네트워크 기반의 개발자 협업 지원 도구들 또한 자신이 도움을 받을 수 있는 타 개발자들을 직접 검색하고 결정해야 하고, 자신에게 맞는 자산을 직접 찾아야 하는 부담이 여전히 발생한다.

3. 컴포넌트 그리드 시스템 아키텍처 설계

본 절에서는 국방 소프트웨어 개발 도메인에서 개발자 친화적인 소프트웨어 재사용 환경을 제공하기 위해 본 연구에서 개발 중인 위키 기반의 컴포넌트 그리드 아키텍처와 그 세부 요소들에 대해 설명한다.

3.1 컴포넌트 그리드 아키텍처

그림 2는 컴포넌트 그리드의 아키텍처를 보여준다. 아키텍처는 크게 세 부분으로 구성 된다: 저장소 계층(Repository layer), 인프라스트럭처 계층(Infrastructure layer), 클라이언트 계층(Client layer). 저장소 계층은 필요한 온톨로지, 컴포넌트, 산출물들이 데이터베이스 화 되어 저장되어 있는 계층을 말한다. 온톨로지 저장소에는 산출물, 아키텍처, 컴포넌트, 요구사항 등의 온톨로지가 저장되어 있고, 컴포넌트 저장소에는 실제 컴포넌트들의 소스코드와 컴포넌트 명세서들이 저장되어 있다. 아키텍처 저장소와 타산출물 저장소에는 컴포넌트가 조합되어 활용될 수 있는 아키텍처 명세와 요구사항 분석서 등의 컴포넌트 개발에 있어서 부가적으로 작성되는 산출물들이 저장된다. 인프라스트럭처 계층은 저장소에 저장되어 있는

온톨로지, 컴포넌트, 기타 산출물 등을 등록, 검색 및 관리 할 수 있도록 지원하여 주는 계층이다. 각 세부 요소들의 역할에 대해서는 다음 절에서 논한다. 마지막으로 클라이언트 계층은 실제 개발자가 재사용 활동을 할 수 있도록 지원해 주는 인터페이스를 제공하는 계층이다. 개발자들은 이 계층을 통해 자산을 등록, 검색 및 관리할 수 있고 재사용 가이드라인을 제공 받을 수 있으며 협력적으로 재사용 자산의 공유 및 증식을 할 수 있다.

3.2 컴포넌트 그리드 아키텍처 세부요소들

본 절에서는 컴포넌트 그리드 아키텍처의 핵심 계층인 인프라스트럭처 계층의 서브 요소들에 대해 설명한다.

- 위키 관리기(Wiki manager): 위키 관리기는 위키 페이지의 생성, 등록, 검색을 지원해 주는 역할을 한다. 재사용 자산 위키 페이지 생성 시 개발자가 입력하여야 하는 템플릿을 제공하고(Template generator), 위키페이지를 통해 시맨틱 태깅이 가능케 도와준다(Tag manager). 개발자가 위키 템플릿을 완성하면 이를 국방 소프트웨어 분류체계에 따라 저장하여(Wiki generator) 타 개발자들과 협력적으로 자산 공유 및 자산 관련 지식 증식을 가능하게 해준다.

- 커뮤니티 관리기(Community manager): 커뮤니티 관리기는 자신과 유사한 요구사항 및 과거 경험을 가지는 개발자들을 소셜 네트워크를 기반으로 검색하여 주고 이러한 개발자들이 커뮤니티를 생성할 수 있도록 지원하여 준다(Community creator). 자신의 요구에 맞는 적절한 타 개발자가 없으면 유사 커뮤니티를 추천하는 기능도 수행한다(Community recommendation engine). 소셜 네트워크 기반의 자산 검색 기법은 5절에서 자세히 논한다.

- 지식 관리기(Knowledge manager): 지식 관리기는 재사용 지식을 관리해 주는 역할을 수행한다. 재사용 지식은 일종의 재사용 가이드라인으로 활용될 수 있으며 타 개발자들의 과거 재사용 경험, 노하우(Know-how), 재사용 문제해결 방법 등을 지식화 하여 저장 및 관리하고 이를 개발자들에게 제공함(Knowledge search engine)으로써 재사용에 대한 부담을 줄이고, 효율적인 재사용을 가능하게 하기 위함이다. 재사용 지식 관련해서는 6절에서 자세히 논한다.

- 질의 관리기(Query manager): 개발자들이 재사용 자산을 검색 할 수 있도록 질의 입력(Input acquisition engine)을 받아 질의를 생성(Query generator)하고 이를 정제(Query refinement)하여 브로커 및 저장소 관리기에 전송한다. 검색 결과를 브로커로부터 받아 이를 포맷팅(Result formatter)하여 개발자들에게 제공하는 역할을 수행한다.

- 브로커(Brokers): 브로커는 질의 관리기에 의해 입력 받은 질의를 처리하여 실제 자산을 검색하는 역할을 수행한다. 브로커는 질의 대상에 따라 산출물 브로커(Artifact broker), 컴포넌트 브로커(Component broker), 아키텍처 브로커(Architecture broker), 서비스 브로커(Service broker)로 나누어진다. 질의하고자 하는 자산에 맞게 시맨틱스 기반의 검색(Semantic gauges)을 통해 질의 결과 값을 추출한다. 브로커 내의 요구사항 추적기(Requirement tracer)는 요구사항 온톨로지와 연계되어 연관 자산을 태깅하는 역할을 수행한다. 요구사항 추적기에 대한 설명은 4절에서 자세히 논한다.

- 명세 관리기(Description manager): 명세 관리기는 재사용 자산을 등록하는 역할을 수행한다. 자산의 등록은 국방 소프트웨어 표준 분류 체계에 따른 시맨틱 모델(Semantic modeler)에 따라 체계적으로 온톨로지에 분류되어 저장되고(Regostration information manager) 실제 자산 파일들도 저장소 관리기에 의해 데이터베이스화 되어 저장된다(Registration engine). 서비스 명세 관리기(Service description manager)는 자산이 컴포넌트의 형태가 아닌 서비스의 형태로 개발되어 졌을 때 이를 등록 하는 역할을 수행한다.

- 저장소 관리기(Repository manager): 저장소 관리기는 저장소 계층에 저장되어 있는 온톨로지(Ontology manager), 컴포넌트, 아키텍처, 기타 산출물(Artifact repository manager) 들을 데이터베이스화 하여 저장하고 데이터베이스에 실제 접근하여 정보를 획득하는 기능을 수행한다. 온톨로지에 URI(Uniform Resource Indicator)를 저장하고 이 URI들이 실제 파일의 경로인 URL(Uniform Resource Locator)과 서로 연계되어 있어 브로커링이 수행된 후 실제 파일에 접근할 수 있도록 지원한다. 데이터베이스 중재기(Database mediator)는 이질적인 데이터베이스를 중재하여 접근할 수 있도록 지원해 주는 역할을 수행한다.
- 관리기능(Admin functions): 관리기능은 사용자의 로그인 정보, 계정관리, 보안관리, 사용자 인터페이스 관리 등을 수행하는 시스템 관리자의 기능을 제공한다.

4. 시맨틱 태깅 기반 요구사항 추적 기법[1]

본 절에서는 시맨틱 태깅을 이용하여 개발자들이 자신의 요구사항에 맞는 적절한 수준의 관련 산출물들을 재사용 할 수 있도록 지원해 주는 추적 기법에 대해 설명한다. 시맨틱 태깅이란 어떠한 대상에 시맨틱 메타데이터의 타입을 기술해 놓는 활동을 말하며 이를 통해 요구사항 과 산출물들 및 자산을 연계하여 재사용 가능하도록 패키지화 할 수 있도록 지원한다. 소프트웨어 자산 추적을 위한 시맨틱 태깅 과정은 먼저 요구사항 온톨로지의 개념을 이용하여 재사용 가능한 자산들을 가지는 요구사항들을 검색하고, 추출된 요구사항을 기반으로 개발자가 원하는 요구사항을 선택한다. 요구사항을 기반으로 관련 있는 산출물들을 검색하여 추출하고, 선택한 요구사항과 유사한 요구사항들을 검색하고 재사용 가능한 자산들을 추출해 낸다. 개발자가 추출된 자산들을 선택하여 본인의 요구사항에 맞도록 수정하고 이를 태깅하여 저장한다. 이렇게 저장된 태그는 타 개발자가 유사한 요구사항을 가질 때 추천되어 태깅된 산출물들이 재사용 될 수 있다. 태깅 추천을 위한 방법에는 크게 명성 기반의 추천과 유사어 집합 기반의 추천 기법이 있다. 명성 기반 추천은 태깅된 특정 태그가 타 개발자에 의해 얼마나 많이 태깅 되었나를 숫자를 기반으로 우선순위를 정해 추천해 주는 방법이고, 유사어 집합 기반의 추천은 충분히 재사용될 태그들이 존재하지 않을 때, 개발자의 입력에 따른 시맨틱 거리를 측정하여 유사성을 측정하여 추천해 주는 기법이다. 시맨틱 태깅 기반의 요구사항 추적 기술은 참고문헌[1]에 보다 자세히 기술되어 있다.

5. 소셜 네트워크 기반 재사용 자산 검색 기법[12]

본 절에서는 유사한 요구사항을 갖거나 과거 재사용 경험이 있는 개발자들을 소셜 네트워크 기반으로 찾아주고 이 개발자들이 보유하고 있는 자산을 재사용 할 수 있도록 지원해 주는 검색 기법에 대해 설명한다. 소셜 네트워크 기반의 재사용 자산 검색을 위해서 본 연구에서는 세가지 관점에 기반 한 자산 검색 모델을 제안한다. 첫 번째로 유사성 관점(Similarity Aspect)은 소셜 네트워크를 구성하는데 있어 가장 중요한 요소로 개발자들이 개발하고자 하는 소프트웨어의 핵심 요구사항과 유사한 요구사항을 가진 소프트웨어를 개발 중에 있거나 개발 했던 개발자들과의 관계 형성을 위한 요소이다. 유사성 관점에는 세부요소로 공통 응용 도메인, 공통 역할, 공통 개발 언어들이 있고 이러한 세부적인 요소들을 계산하여 개발자들 간의 유사성을 측정하게 된다.

두 번째는 친밀성 관점(Familarity Aspect)인데 이는 개발자들 간에 서로 얼마나 잘 알고 있는지를 나타내는 지표로 재사용 관련하여 서로 얼마나 상호작용을 있는지를 나타내는 요소이다. 친밀성은 재사용 관계와 재사용을 위한 상호작용을 측정함으로써 계산될 수 있다. 재사용 관계는 다른 개발자의 자산을 본인이 재사용한 적이 있는지 여부를 나타내고, 재사용 상호작용은 재사용 활동을 위해 서로 주고받은 의사소통의 정도를 나타낸다. 마지막으로 친화성은 다른 개발자에 대한 호감 정도를 나타내는 요소로 소유하고 있는 재사용 가능한 자산의 수, 자산의 재사용 횟수, 평판에 따라 측정될 수 있다. 이러한 세 가지 요소와 세부 요소들에 따라 그림 3과 같이 소셜 네트워크 기반의 커뮤니티 추천 모델을 개발하였다.

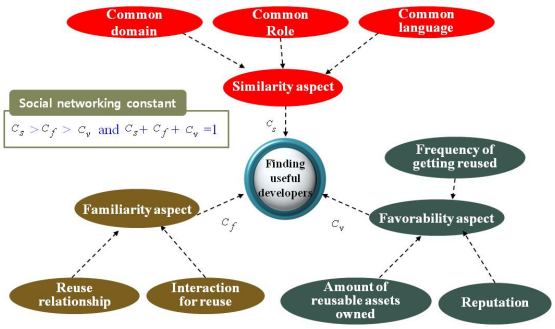

그림 3 소셜 네트워크 기반의 커뮤니티 추천 모델 [12]

C_s(Similarity Constant)는 유사성의 중요도를, C_f(Familarity Constant)는 친밀성의 중요도를 나타내고 C_V(Favorability Constant)는 친화성의 중요도를 나타내며 그 합은 '1'이다. 중요도에 가중치 및 우선순위를 부여하는 방법 및 랭킹을 계산하는 수식은 참고문헌 [13]에서 자세히 기술하고 있다.

6. 소프트웨어 재사용 지식 표현 모델 [13]

본 절에서는 재사용의 가이드라인 제공 및 재사용함에 있어 직면할 수 있는 문제점을 해결해 주기 위한 재사용 지식에 관해 설명하고 재사용 지식 표현 모델에 대해 설명한다. 본 연구에서는 소프트웨어 재사용 지식을 "재사용과 관련된 개인이나 기관의 경험, 혹은 다른 개발자에게 도움을 줄 수 있는 유용한 히스토리 정보"라고 정의하였다. 그림 4는 재사용 지식을 정형화하여 표현하기 위한 온톨로지 기반의 표현 모델로서, 중요 구성 요소들에 대한 설명은 아래와 같다[14, 15, 16].

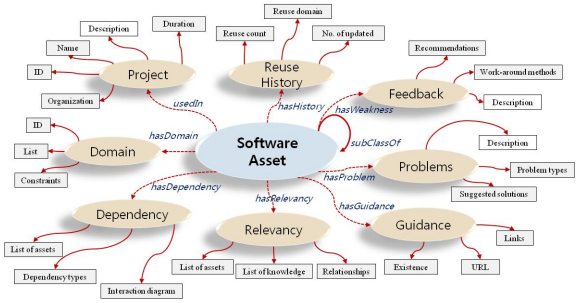

그림 4. 소프트웨어 재사용 지식 표현 모델 [13]

- 프로젝트(Project): 자산이 개발된 프로젝트의 정보를 표현한다. 개발자들은 자신이 현재 개발하고자 하는 소프트웨어와 유사한 기존 프로젝트 정보에서 재사용 자산들과 지식을 얻을 수 있다. 기능적 요구사항 관점에서 중요한 정보를 제공할 수 있다.
- 도메인(Domain): 자산이 개발되어 활용되는 응용 도메인을 나타낸다. 도메인 정보는 공통정의, 공통 개념, 공통 제약사항 등의 도메인 정보를 제공할 수 있다.
- 의존성(Dependancy): 의존성은 재사용할 자산이 다른 특정 자산들과 통합되어 사용되어야 한다는 정보를 나타낸다. 이를 통해 개발자들은 필요한 타 자산을 쉽게 파악할 수 있다.

- 연관성(Relevancy): 자산과 관련된 산출물들의 정보를 나타낸다. 요구사항 분석서, 설계서 등의 부가 산출물들의 정보를 제공함으로써 개발자들이 재사용하기 용이하게 도와준다.
- 재사용 문제 해결(Reuse Problem Solving): 소프트웨어 재사용은 크게 문제 해결 활동으로 볼 수 있다. 개발자들은 타 개발자가 개발한 자산을 재사용 시 보통 어려움과 문제점에 직면하게 된다. 다른 개발자들이 재사용 활동에 있어서 겪었던 문제점과 해결 경험을 지식 화하여 공유한다면 이는 다른 개발자들의 재사용 증대에 기여를 할 수 있다.
- 피드백(Feedback): 재사용함에 있어서 자산의 평가 정보를 나타낸다. 자산의 재사용 용이성, 어려운 점, 개선 방향 등을 지식 화하여 타 개발자들과 공유할 수 있게 지원한다.
- 지도(Guidance): 명백한 재사용 설명이나 튜토리얼 등의 문서가 있는지 여부를 나타낸다. 재사용할 때 수행해야 하는 작업들, 재사용의 적절성 여부를 파악하기 쉽게 도와준다.
- 재사용 히스토리(Reuse History): 자산의 재사용 역사를 보여준다. 재사용 되었던 도메인, 재사용 횟수 등을 지식 화하여 제공함으로써 개발자들이 얼마나 신뢰성 있는 자산인지 어떠한 분야에 재사용 되었었는지를 쉽게 알 수 있도록 지원한다.

7. 구현된 컴포넌트 그리드 프로토타입

본 절에서는 구현된 컴포넌트 그리드 시스템의 프로토타입에 대해 설명한다. 그림 5는 자산 관리를 위한 컴포넌트 그리드 환경 중 컴포넌트의 등록 화면을 보여준다. 컴포넌트 파일 업로드 기능을 제공하고, 컴포넌트 명세를 위해 도메인, 기능적 요구사항, 비기능적 요구사항 등의 명세 정보를 개발자들이 직접 입력하는 것이 아니라 타입들을 선택할 수 있도록 지원하여 줌으로써 컴포넌트 등록 및 명세를 위한 개발자들의 부담을 줄여준다.

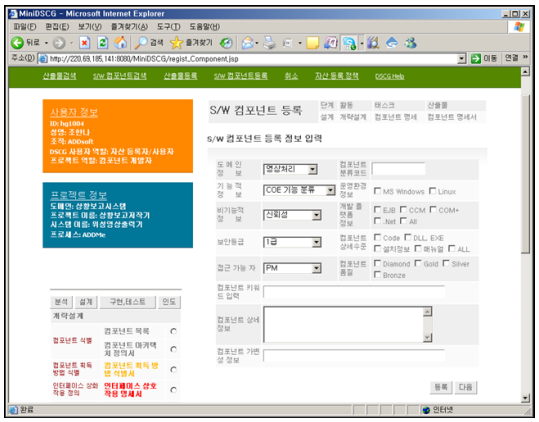

그림 5. 웹기반 재사용 자산관리 환경

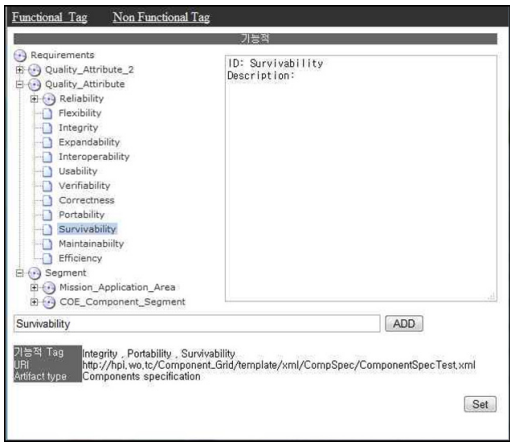

그림 6. 태깅 기반 요구사항 추적기 프로토타입

그림 6은 4절에서 설명하였던 태깅 기반 요구사항 추적기의 구현된 프로토타입을 보여준다. 그림 5의 웹기반 자산관리 환경에서 태깅을 원할 때 사용되는 인터페이스이다. 개발자가 특정 자산 페이지에서 태그 생성 버튼을 누르면 그림의 좌측에서처럼 미리 정의된 요구사항 온톨로지 클래스 구조를 나타내 주고 적절한 요구사항 클래스를 선택하여 태그를 생성할 수 있으며, 적절한 클래스가 없을 경우에는 새로운 클래스를 정의

하고 태그의 메타데이터를 명세하여 새로운 태그를 생성할 수 있다. 이러한 태그들을 통해서 연관 재사용 자산을 요구사항에 맞게 추적할 수 있고 관련 산출물을 찾는 노력을 줄일 수 있다.

그림 7은 5절에서 설명하였던 소셜 네트워크 기반 재사용 자산 검색을 위한 구현된 프로토타입을 보여준다. 좌측의 레이더 뷰(Radar view)는 자신과 유사성, 친밀성, 친화성을 가지는 개발자들의 사진을 보여주고 사진을 선택 시, 그 개발자의 정보를 보여준다.

개발자를 선택하면 개발자가 보유하고 있는 재사용 자산에 접근할 수 있다. 가운데 본인을 중심으로 가까울수록 자신의 재사용 활동에 도움을 줄 수 있는 개발자임을 나타내고, 이 거리는 앞서 설명한 세 가지 요소의 세부 인자들을 측정한 결과 값을 기반으로 한다. 우측 에는 세부 인자들을 기반으로 그 우선순위를 다르게 하여 개발자들을 나타낼 수 있도록 지원하기 위한 버튼 들이고, 유사성에 최고 가중치를 두고 타 개발자들을 찾아봐도 유용한 정보를 얻을 수 없을 때 사용할 수 있다. 이렇게 자산 기반이 아닌 소셜 네트워크 기반의 재사용을 지원함으로써 유사한 경험과 요구사항을 가지는 개발자들 간의 의사교환을 원활히 할 수가 있고, 서로 보유하고 있는 자산을 교류할 수 있다. 또한, 커뮤니티를 통해 자유롭게 의사소통을 할 수가 있다.

그림 8은 구현된 컴포넌트 그리드 위키의 메인 페이지를 보여준다. 위키 메인 페이지는 앞서 설명한 웹기반 컴포넌트 관리 환경과 연동되어 실행되고 컴포넌트 관리 환경에 등록 되어 있는 재사용 자산의 현황을 보여준다. 그림의 좌측 윗부분의 원형 그래프는 현재 등록되어 있는 자산의 현황을 국방 소프트웨어 분류체계에 따라 보여주고 있고, 그 아래에는 소셜 네트워크 기반의 재사용 자산 검색 기능이 통합되어 있다. 오른쪽 부분에는 국방 소프트웨어 분류체계에 따라 등록되어 있는 컴포넌트들의 리스트를 보여주고, 컴포넌트를 선택하면 컴포넌트에 대한 자산 위키 페이지로 접근할 수 있다. 재사용 위키 메인 페이지의 목적은 재사용 가능한 자산의 현황을 보여줌으로써 개발자들로 하여금 자산 접근을 용이하게 해주는데 있다.

그림 7. 소셜 네트워크 기반의 자산 검색 환경[12]

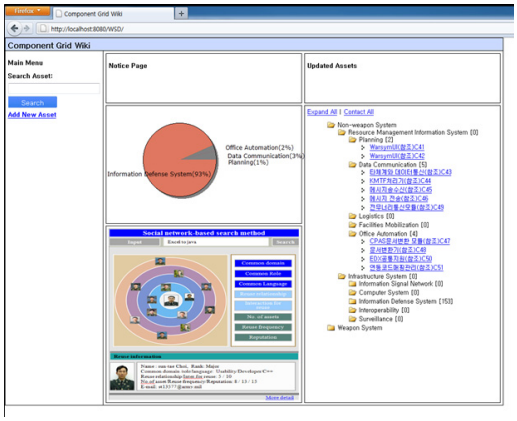

그림 8. 구현된 재사용 위키의 메인 페이지

그림 9는 구현된 재사용 위키의 자산 페이지를 보여준다. 그림의 윗부분은 자산의 명세에 대한 내용인데 웹기반 자산 관리기를 이용하여 등록된 컴포넌트의 간략한 명세를 나타내어 준다. 도메인

정보, 프로젝트 정보, 연관 자산, 인터페이스 등의 정보를 한눈에 파악할 수 있도록 지원하여 준다. 그림의 아랫부분은 6절에서 설명한 소프트웨어 재사용 지식 표현 모델에 따라 기술되어 있는 재사용 지식을 보여준다. 재사용 지식에서 가장 중요한 부분은 재사용 문제 해결 부분인데, 관련 연구 및 설문조사 활동을 토대로 일반적인 소프트웨어 재사용에 있어서 발생할 수 있는 기술적인 문제들의 타입을 정의하였다. 타입은 크게 신뢰성(Reliability), 호환성(Compatibility), 상호운용성(Interoperability), 비완전성(Incompleteness), 가시화(Visualization) 문제로 나눌 수 있고 각 문제에 따른 세부타입도 정의하였다[14].

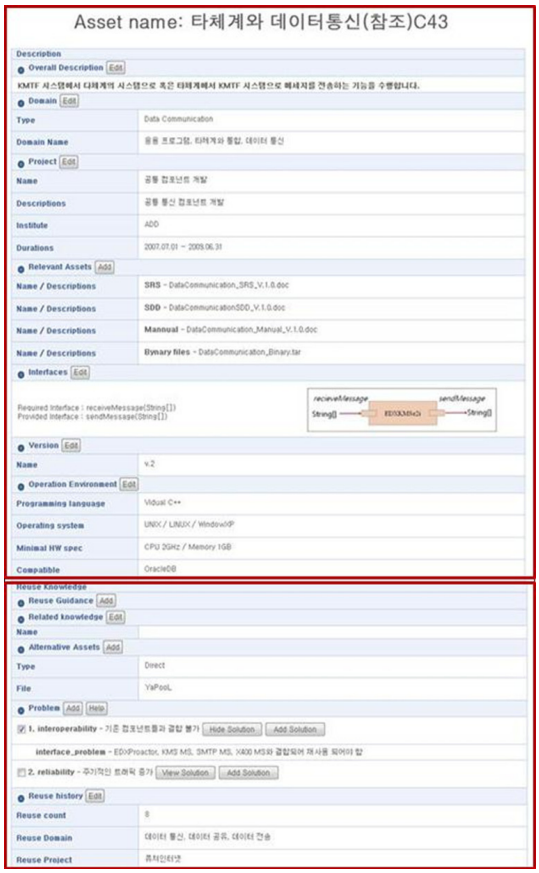

그림 9. 구현된 재사용 위키의 자산 및 지식 페이지

이렇게 타입을 정의한 이유는 개발자들이 자신이 어떠한 재사용 문제에 직면하였는지를 자연어로 기술하게 하기 보다는 일반적인 타입 중에서 하나를 선택하고 세부 문제도 선택하게 함으로써 재사용 문제 이해 및 등록을 용이하게 하기 위함이다. 현재 이러한 타입에 따른 문제해결 지식 추천 방법을 개발 중에 있고, 지식 추천을 위한 정형화된 지식의 표현법 및 문제 해결 추천 룰(Rule)들을 정의하고 있다. 재사용 지식이 증식되고 정제됨에 따라 개발자들은 어떻게 재사용 하여야 하는지 가이드라인을 얻을 수 있어 재사용에 대한 부담을 감소시키고, 지식의 정제 및 증식은 위키 인터페이스를 통해 여러 개발자들에 의해 협력적으로 이루어지게 되므로 점차 자산 및 지식의 품질과 신뢰성이 좋아지게 된다.

8. 결론 및 향후연구

국방 소프트웨어 개발 분야는 그 응용 도메인이 다양하고 규모가 커서 소프트웨어 재사용의 효과를 많이 볼 수 있고, 재사용이 꼭 필요한 분야이다. 그러나 국방 소프트웨어 개발 실태 및 개발자들이 겪는 부담 등의 문제점들로 인해 실질적으로는 재사용이 잘 이루어지지 않고 있는 실정이다. 본 연구에서는 이 문제점들의 원인을 분석하고 그 해결방안으로 전사적 군 소프트웨어 자산의 재사용을 지원해 줄 수 있는 개발자 친화적인 컴포넌트 그리드 시스템을 개발 중에 있다. 효율적인 의사소통 방법 제공을 위해 소셜 네트워크 기반의 자산 검색 및 커뮤니티 지원 기법을 개발하였고, 소프트웨어 자산의 상호 교류 및 지원을 위해 웹 기반의 자산 관리 환경을 개발하였다. 자산 재사용에 대한 가이드라인을 제공하기 위해 소프트웨어 재사용 지식을 정의하고 온톨로지 기반의 재사용 지식 표현 모델을 개발하였으며, 위키 기반의 협력적 지식 공유 및 축적 환경을 구현하였다.

개발자들의 재사용에 대한 부담을 줄여주기 위하여 시맨틱 태깅 기반의 요구사항 추적 기술, 재사용 문제해결 지식 제공 방법, 소셜 네트워크 기반의 유사성 검색을 통한 도움 받을 수 있는 개발자 및 자산 추천 기법 또한 개발하였다. 이러한 재사용 환경은 사용하기 쉽고 친숙한 웹과 위키 기반으로 구현되었으므로 도구 사용에 대한 개발자들의 부담 또한 감소하여 줄 수 있다. 본 연구의 결과물인 컴포넌트 그리드 시스템의 개발은 전군적 소프트웨어 재사용을 가능케 하고 참여적 협력적인 재사용 자산 및 지식의 정제 및 증식을 가능하게 지원하여 결과적으로 국방 소프트웨어의 품질과 신뢰성을 향상시켜 재사용성을 증대할 수 있으리라 기대한다.

현재 시맨틱 태깅 기반의 요구사항 추적 기술과 소셜 네트워크 기반의 자산 검색 기술을 고도화하기 위한 전략 및 방법을 위한 연구를 진행 중에 있고, 재사용 문제 해결을 위한 룰의 정의 및 룰 기반 문제 해결방안 추천 기법을 연구 중에 있다. 이러한 연구들을 바탕으로 실제 국방 도메인에 쓰일 수 있도록 환경의 통합 및 배포, 평가 계획을 수립 중에 있다.

참 고 문 헌

[1] 구형민, 조혜경, 최선태, 고인영, “국방 소프트웨어 자산의 재사용성 증대를 위한 웹 2.0 기반의 재사용 지원 환경”, 소프트웨어공학소사이어티 논문지, 제22권 제1호, 2009년 3월.

[2] Stephan J. Andriole, “Business Impact of Web 2.0 Technologies”, Communications of the ACM, 2010.

[3] Ryan Gerard and et al., “The Software Reuse Working Group: A Case Studying in Fostering Reuse”, NASA, IEEE, 2007.

[4] Bruno Antunes and et al., “SRS: Software Reuse System based on Semantic Web”, Proceeding of the 3rd International Workshop on Semantic Web Enabled Software Engineering, 2007.

[5] Tom Heath and Enrico Motta, “Revyu.com: A Reviewing and Rating Site for the Web of Data”, ISCW/ASWC, 2007.

[6] Bjorn Decker and et al., “Self-organized Reuse of Software Engineering Knowledge Supported by Semantic Wikis”, Proceedings of Workshop on Semantic Web Enabled Software Engineering (SWESE), IEEE, November,2005.

[7] H. Happel and S. Seedorf, “Ontobrowse: A Semantic Wiki for Sharing Knowledge about Software Architectures”, In Proceedings of the 19th International Conference on Software Engineering and Knowledge Engineering (SEKE), 2007.

[8] Devpia web site (http://www.devpia.co.kr)

[9] Devmento web site (http://www.devmento.co.kr)

[10] Gosu web site (http://www.gosu.co.kr)

[11] Stackoverflow web site (http://www.stack overflow. co.kr)

[12] 최선태, 구형민, 고인영, “소셜 네트워크 기반의 소프트웨어 재사용 자산 검색 기법”, 정보과학회 논문지: 소프트웨어 및 응용, 제38권, 제8호, 2011년 8월.

[13] 구형민, 고인영, 2011 한국 소프트웨어공학 기술 합동 워크샵, 제9권, 제1호, 2011년 8월.

[14] Hing Yan Lee and et al., “Software Engineering Knowledge for Software Reuse”, IEEE, 1993.

[15] Eric Ras and Sebastian Weber, “Software Organization Platform: Integrating Organization and Individual Learning”, Proceedings of Wiki4SE ’09, ICSE, 2009.

[16] Frank McCarey and et al., “Knowledge Reuse for Software Reuse”, Web Intelligent and Agent Systems: An International journal 6, 2008.

저 자 소 개

고 인 영

1990년 서강대학교 전산학과 학사
1992년 서강대학교 전산학과 석사
2003년 미국 Univ. of Southern California박사
1993년~1996년 공군사관학교 교관 (전임강사)
1997년 미국 USC Information Science Institute (ISI) 연구 조교
2003년 미국 USC ISI Postdoctoral Research Associate
2004년~현재 한국과학기술원 전산학과 부교수

구 형 민

2004년 금오공과대학교 컴퓨터공학과 학사
2007년 한국과학기술원 전산학과 석사
2007년 한국전자통신연구원 위촉연구원
2007~현재 한국과학기술원 전산학과 박사과정

회원 가입 안내 및 회비 납부 요령

한국정보과학회 소프트웨어공학소사이어티는 회원 여러분에게 유익한 정보를 제공해 드리기 위하여 보다 충실한 내용의 논문지 발간 배포, 그리고 국제·국내 학술발표회 및 초청강연회와 단기강좌 등의 여러 가지 사업들을 추진하고 있습니다.

소프트웨어공학 소사이어티의 가입을 통해 정보 및 기술 교류, 그리고 인적 네트워크의 구성에 참여하시기를 기대합니다. 회원 가입을 위하여 아래의 회비 안내를 참고하시어 회비를 납부하시고, 다음 쪽의 입회원서를 작성하시어 아래 소프트웨어공학소사이어티 주소로 보내주시거나 팩스 또는 이메일을 통해 보내어 주시기 바랍니다.

한국정보과학회 소프트웨어공학소사이어티 연락처는 아래와 같습니다.

◈ 소프트웨어공학소사이어티

주 소 : (우) 121-742 서울시 마포구 신수동 1번지, 서강대학교 신과학관 202호
한국정보과학회 소프트웨어공학소사이어티 박수용

전 화 : (02) 705-8928

팩 스 : (02) 704-8273

전자우편 : sypark@sogang.ac.kr (박수용교수), bjlee@uos.ac.kr (이병정교수)

홈페이지 : http://www.sigse-kiss.or.kr/

◈ 회비 안내

회원구분	• 학생회원 : IT 분야 학과 또는 관심 있는 학생 • 정회원, 종신회원 : IT 분야 종사자
가입비	학생회원, 정회원 : 20,000원, 종신회원 : 200,000원
년회비	학생회원, 정회원 : 20,000원

- 회비납부 방법

(1) 무통장입금 또는 계좌이체 후 입회원서 발송
: 계좌 번호 : 제일은행 150-20-358028 (이병정)

(2) 소사이어티 주관 학술행사 개최시, 행사장 당일 가입 및 납부 가능

개인회원용 입회원서

회원구분	학생회원 ( ) 정회원( ) 종신회원( )				
성명	한글		생년월일		
	영문				
연락처	직장전화		휴대전화		
	e-mail				
주소	직장명/부서		직급		
	직장주소	(우)			
학력	학사	년 월 - 년 월	대학교	과	
	석사	년 월 - 년 월	대학원	과	
	박사	년 월 - 년 월	대학원	과	
관심분야					

본인은 한국정보과학회 소프트웨어공학소사이어티의 취지에 찬성하여 회원으로 가입하고자 이에 입회원서를 제출합니다.

년 월 일

신 청 인: (인)

한국정보과학회 소프트웨어공학 소사이어티 회장 귀하

논문지 논문 모집 (Call for Papers)

한국정보과학회 소프트웨어공학 소사이어티에서는 매년 4회에 걸쳐 '소프트웨어공학 소사이어티 논문지' 를 발간하고 있습니다. 이 논문지에는 소프트웨어공학 전반에 걸친 연구논문과 산업계 논문을 게재해 오고 있습니다. 다음과 같은 소프트웨어공학 주제에 관련된 논문을 모집하고 있으니 학계와 산업계의 여러분들의 적극적인 논문투고를 바랍니다.

◈ 논문 주제

- ○ 소프트웨어 설계 및 아키텍처
- ○ 소프트웨어 재사용 및 프로덕트라인
- ○ 요구공학
- ○ 소프트웨어 품질 및 테스트
- ○ 관리 및 프로세스
- ○ 소프트웨어 정형 기법
- ○ 서비스기반 소프트웨어 개발
- ○ 임베디드, 모바일, 웹기반 소프트웨어 개발
- ○ 기타 소프트웨어 응용 (국방, 자동차, 조선 등의 분야)

◈ 논문심사

- ○ 투고된 논문은 편집위원회에서 심사 선정하며, 필요 시 외부 심사위원을 위촉하여 심사를 합니다. 제출된 논문은 반환하지 않습니다.
- ○ 심사료 및 게재료: 없음

◈ 논문 제출

- ○ 한국정보과학회 논문지 투고 양식(www.kiise.or.kr)을 사용하며, 논문의 분량은 10장으로 제한합니다.
- ○ 논문지 투고규정에 따라 작성된 심사용 논문파일과 함께 논문제목, 저자, 소속, 초록을 편집위원장 또는 편집위원에게 이메일로 송부하시기 바랍니다.

◈ 문의처 (편집위원회)

- ○ 편집이사 : 고인영 교수 (KAIST, 042-350-3547, iko@kaist.ac.kr)
- ○ 편집이사 : 윤회진 교수 (협성대학교, 031-299-0841, hjyoon@uhs.ac.kr)
- ○ 편집위원 : 강성원 교수 (KAIST, 042-350-3512, sungwon.kang@kaist.ac.kr)
- ○ 편집위원 : 김문주 교수 (KAIST, 042-350-3543, moonzoo@cs.kaist.ac.kr)
- ○ 편집위원 : 김정아 교수 (관동대학교, 033-649-7801, clara@kd.ac.kr)
- ○ 편집위원 : 이우진 교수 (경북대학교, 053-950-6378, woojin@knu.ac.kr)
- ○ 편집위원 : 이찬근 교수 (중앙대학교, 02-820-5829, cglee@cau.ac.kr)
- ○ 편집위원 : 이 근 박사 (삼성전자, 031-277-7323, gskeun@gmail.com)

투 고 요 령

1. 소프트웨어공학소사이어티 논문지에 실리는 원고는 주제 논문, 일반 논문, 산업체 기고 등으로 구분하며 다음과 같은 분야에 대하여 모집한다.
   가. 소프트웨어공학 및 그 응용분야에 대한 연구결과
   나. 강좌 및 관련 교육사항 소개 (목적, 과정, 일정, 대상, 특징)
   다. 소프트웨어 도구 및 방법론 소개 (가격, 특징, 종류, 적용사례)
   라. 소프트웨어 산업에 대한 학계, 업계의 주요 관심사
   마. 기타 관련 사항

2. 투고자는 원칙적으로 본 소사이어티의 회원으로 한다. 다만 공동 또는 초청 기고자는 예외로 한다.

3. 논문은 원칙적으로 한글로 작성한다.

4. 원고는 '한글'이나 'MS 워드'로 작성하며 논문 양식은 한국정보과학회 온라인 논문 투고 사이트(http://www.kiise.or.kr/)에서 받아 사용한다.

5. 논문은 그림과 표를 포함하여 10장 내외로 한다.

6. 작성된 논문은 논문파일과 함께 논문제목, 저자, 소속, 초록을 편집위원장 또는 특집 주제 담당 편집위원에게 이메일로 제출한다.

7. 기타 자세한 사항은 한국정보과학회 논문지 투고 요령을 따른다.

한국정보과학회 소프트웨어공학소사이어티 임원명단

직책		성명	소속
회 장		박 수 용	서강대학교
부 회 장		권 기 현	경기대학교
부 회 장		민 경 호	LG전자
부 회 장		전 진 옥	비트컴퓨터
부 회 장		최 병 주	이화여자대학교
부 회 장		한 혁 수	상명대학교
감 사		이 긍 해	항공대학교
		차 성 덕	고려대학교
자 문 위 원 회		강 교 철	포항대학교
		권 용 래	KAIST
		배 두 환	KAIST
		성 기 수	KISTI 고문
		신 규 상	ETRI
		양 승 민	숭실대학교
		우 치 수	서울대학교
		이 경 환	중앙대학교
		이 단 형	KAIST
		정 기 원	숭실대학교
		황 선 명	대전대학교
협 력 기 관 위 원 회		권 경 룡	국방품질기술원
		박 찬 규	국방SW산학연합회
		신 석 규	SW시험인증센터
		유 법 민	지식경제부
		윤 태 권	SW기술진흥협회
		이 상 은	한국소프트웨어진흥원
운영위원회	**운 영 위 원 장**	권 기 현	경기대학교
	총 무 이 사	이 병 정	서울시립대학교
	기 획 이 사	백 종 문	KAIST
	조 직 이 사	이 관 우	한성대학교
	학 술 이 사	홍 장 의	충북대학교
	편 집 이 사	고 인 영	KAIST
	편 집 이 사	윤 회 진	협성대학교
	협 력 이 사	인 호	고려대학교
	홍 보 이 사	이 정 원	아주대학교
기술위원회	**기 술 위 원 장**	최 병 주	이화여자대학교
	요구공학분과장	박 수 진	서강대학교
	아키텍쳐분과장	조 은 숙	서일대학
	재사용분과장	이 관 우	한성대학교
	정형화분과장	김 문 주	KAIST
	테스팅분과장	김 영 철	홍익대학교
	프로세스분과장	이 세 영	NIPA

직책		성명	소속
	고신뢰성분과장	배현섭	슈어소프트테크
	재공학분과장	김진태	SEEG
	실시간분과장	이찬근	중앙대학교
산학위원회	**산학위원회장**	전진옥	비트컴퓨터
	자동차분과위원장	유영수	오토에버시스템㈜
	국방분과위원장	윤희병	국방대학원
	금융분과위원장	김정아	관동대학교
	전자분과위원장	이근	삼성전자
	의료분과위원장	최호진	KAIST
	전력분과위원장	이장수	한국원자력 연구소, MMIS팀 책임연구원
이사		강성원	KAIST
이사		계승교	삼성SDS
이사		권원일	STA컨설팅
이사		남영광	연세대학교
이사		김문주	KAIST
이사		김상기	현대자동차
이사		김영철	홍익대학교
이사		김정아	관동대학교
이사		민상윤	솔루션링크
이사		박복남	핸디피엠지
이사		백종문	KAIST
이사		손세창	인천공항공사
이사		손진규	삼성탈레스
이사		양상욱	KAI
이사		염근혁	부산대학교
이사		오재원	카톨릭대학교
이사		유준범	건국대학교
이사		윤형진	케피코
이사		윤희병	국방대학원
이사		이근	삼성전자
이사		이병걸	서울여자대학교
이사		이성남	방위사업청
이사		이우복	삼성전자
이사		이우진	경북대학교
이사		이은주	경북대학교
이사		이정원	아주대학교
이사		이해서	솔루션링크
이사		장주수	모아소프트
이사		정연대	N3SOFT
이사		정인상	한성대학교
이사		조병인	국방과학연구소
이사		조상윤	다한테크
이사		최승훈	덕성여자대학교
이사		최종무	단국대학교
이사		현창문	탐라대학교

2010-2011 소프트웨어공학소사이어티 논문지 편집위원회

소프트웨어공학소사이어티 논문지 제23권 제4호 (통권 88호)

발 행 일 ॥ 2010년 12월 31일

발 행 인 ॥ 박수용
편 집 인 ॥ 고인영

발 행 처 ॥ 사단법인 한국정보과학회 소프트웨어공학소사이어티
연 락 처 ॥ 서울특별시 마포구 신수동 1번지, 서강대학교 신과학관 202호
전 화 : 02-705-8928, 팩 스 : 02-704-8273
홈페이지 : http://www.sigse-kiss.or.kr/

인 쇄 처 ॥ (주)참기획 (전화 : 042-861-6380, 팩스 : 042-861-6381)