트랜잭션의 시점, 크기 및 개발자의 숙련도를 고려한 논리적커플링 측정기법

(A Logical Coupling Measurement Method Based on Transaction Time, Size and Expertise of Developer)

심 빈 구 † 김 진 태 †† 박 수 용 †††
(Bingu Shim) (Jintae Kim) (Sooyong Park)

요 약 소프트웨어의 수명이 길어짐에 따라, 소프트웨어 유지보수비용을 줄이기 위한 기법에 대한 연구의 중요도가 높아지고 있다. 소프트웨어 엔티티들간의 커플링(Coupling)정보는 소스코드 분석 및 리팩토링등의 소프트웨어 유지보수에 활동에서 의사결정시 정량적인 근거자료로 유용하게 사용되고 있다. 논리적 커플링(Logical Coupling)은 소프트웨어가 진화하는 과정에서 관측된 엔티티들간의 관계성 정도를 나타낸다. 논리적커플링이 처음 소개된 이후로, 커플링의 단위를 상세화 하는 연구가 발표됨으로써, 큰 단위의 엔티티간 논리적커플링으로는 설명할 수 없는 소프트웨어의 특성들이 설명되었다. 하지만, 기존 연구는 프로젝트 구성원들의 성향 및 프로젝트의 특성을 고려하지 못 하였다. 본 연구는 버전관리시스템의 변경기록에 담겨있는 프로젝트 구성원들의 성향과 프로젝트 자체의 특징을 고려한 다중관점 기반의 논리적커플링 측정기법을 제안하고, 오픈소스프로젝트를 통해 제안하는 기법을 검증하였다.

키워드 : 소프트웨어 유지보수, 논리적커플링

Abstract The priority of software maintenance researches has been increasing, since the lengths of software lifecycle are more increasing. Measuring couplings among software entities provides a good quantitative source for analyzing source code and point out candidate refactoring positions. Logical-coupling measures how strongly two software entities are related with each other from the evolutionary point of view. The researches on logical-coupling have been focusing on improving the correctness and explaining more aspects that are hiding by measuring logical-coupling among finer-grained entities. However, existing researches on logical-coupling fails to consider characteristics of developers and projects reflected in transactions. The research proposes a logical-coupling measurement method based on transaction time, size and expertise of developer to improve the correctness by considering characteristics of developers and projects reflected in transactions. The method has been validated by applying it to three open-source projects.

Key words : Software maintenance, Logical-coupling

· 본 연구는 지식경제부 및 정보통신산업진흥원의 대학 IT연구센터 지원사업의 연구결과로 수행되었음(NIPA-2009-(C1090-0903-0004))

† 학생회원 : 서강대학교 컴퓨터공학과
lanore@gmail.com
†† 정 회 원 : 서강대학교 컴퓨터공학과 교수
canon.kim@gmail.com
††† 정 회 원 : 서강대학교 컴퓨터공학과 교수
sypark@sogang.ac.kr

논문접수 : 2009년 7월 6일
심사완료 : 2009년 9월 14일

정보과학회논문지: 소프트웨어 및 응용 제36권 제11호(2009.11)

1. 서 론

소프트웨어의 수명이 길어짐에 따라, 소프트웨어의 유지보수 활동이 차지하는 비중이 높아지고 있다. 이에 따라, 소프트웨어 유지보수비용을 줄이기 위한 기법에 대한 연구의 중요도 역시 높아지고 있다. 소프트웨어 엔티티들간의 커플링(Coupling)정보는 소스코드 분석 및 리팩토링 등 소프트웨어 유지보수에 활동에서 의사결정시 정량적인 근거자료로 유용하게 사용되고 있다[1]. 1974년 Stevens에 의해 소프트웨어를 구성하는 모듈들 간의 커플링 개념이 처음 소개된 뒤로, 다양한 관점에서의 커플링에 대한 연구가 진행되었다. Arisholm는 객체지향

의 다형성과, 동적바인딩(Dynamic Binding)등 소스코드 분석만으로 측정할 수 없는 커플링 수치를 측정하기 위해 다이나믹커플링(Dynamic Coupling) 개념을 소개하였다[2]. Poshyvanyk은 소스코드의 함수명, 변수명, 그리고 코멘트에 포함되어 있는 의미적 정보를 활용하여 소프트웨어 엔티티들 간의 의미적 관련성 정도를 측정하는 개념적 커플링(Conceptual Coupling)을 소개하였다[3].

논리적커플링(Logical Coupling)은 소프트웨어 엔티티들의 변경기록을 기반으로 진화적(evolutionary) 관점에서 소프트웨어 엔티티들간의 관련성 정도를 설명하기 위해 1998년 Gall에 의해 소개된 개념이다[2]. 논리적커플링은 프로그램이 진화하는 과정에서 엔티티쌍이 얼마나 자주, 짧은 범위의 시간 안에서 같이 변경되는지의 정도를 나타낸다. 어떤 엔티티쌍의 논리적커플링이 높다면, 이들은 과거에 같이 변경되는 빈도수가 높았고, 미래에 같이 변경될 확률이 높다고 말할 수 있다. 즉, 논리적커플링은 변경 히스토리를 기반으로 과거에 자주 변경되었고, 앞으로도 변경될 확률이 높은 엔티티들간의 상관관계를 설명함으로써, 소프트웨어의 유지보수에 들어가는 비용을 필요한 곳에 집중할 수 있게 하는 정량적 근거를 제공한다.

Gall에 의해 모듈단위의 논리적커플링개념이 처음 소개된 이후로, 관련 연구들은 논리적커플링의 단위를 상세화 함으로써, 큰 단위 소프트웨어 엔티티간의 논리적 커플링으로는 설명할 수 없는 소프트웨어의 특성들을 설명하였다[4-6]. 하지만, 이들 연구는 프로젝트 구성원들의 성향 및 프로젝트의 특징을 고려하지 않고 논리적 커플링을 추출하였다. 본 연구는 버전관리시스템의 변경기록에 담겨있는 프로젝트 구성원들의 성향과 프로젝트 자체의 특징을 고려한 다중관점 기반의 논리적커플링 측정기법을 제안하고, 오픈소스프로젝트를 통해 제안하는 기법을 검증하였다.

본 논문의 구성은 다음과 같다. 1장의 서론에 이어 2장에서는 본 논문이 주제로 하는 논리적커플링에 대해서 기술하고 논리적커플링 연구의 기반 개념인 버전컨트롤 시스템과 정적커플링에 대해서 설명한다. 3장에서는 논리적커플링에 대한 기존 연구들에 대해서 기술하고 4장에서는 본 논문에서 제안하는 논리적커플링 측정기법에 대해서 자세히 설명한다. 제안된 기법은 5장에서 오픈소스프로젝트에 적용하여 검증하고, 6장에서는 논문에서 구현된 도구를 설명한다. 마지막으로 7장에서 결론을 도출한다.

2. 배경

본 장에서는 본 논문에서 제안하는 기법이 기반으로 하고 있는 주제인 버전컨트롤시스템의 일반적인 동작원리 및 특성들을 기술하고, 정적커플링이 무엇인지에 대한 배경정보를 기술한다.

2.1 버전컨트롤 시스템

버전컨트롤시스템은 소프트웨어 개발에 수반되는 산출물의 버전을 관리하며, 산출물들에 접근권한을 컨트롤하는 서비스를 제공하는 시스템을 총칭한다[7]. 버전컨트롤시스템은 일반적으로 산출물들의 변경을 숫자나, 코드로 표현되는 Revision이라 불리는 식별자를 기반으로 관리한다. 즉, 하나의 파일에 대해서 변경이 이루어질 때마다 그 변경을 식별할 수 있는 유일한 숫자나 코드로 표현되는 Revision값을 할당하고, 차후에 Revision을 이용하여 변경을 조회한다.

버전컨트롤시스템이 수반되는 개발환경에서의 변경은 체크아웃(Check out), 파일의 수정, 그리고 커밋(Commit)의 3단계로 이루어진다. 체크아웃단계는 로컬 저장소에 있는 파일보다 최신버전이 중앙저장소에 있을 경우에, 로컬 저장소의 파일을 중앙저장소에 있는 최신버전의 파일로 업데이트하는 단계이다. 커밋단계는 수정단계에서 개발자에 의해 수정된 파일들을 중앙 서버에 업로드 하여 변경을 반영하고 반영된 변경에 대해 Revision을 할당 받는 단계이다.

2.2 정적 커플링

소프트웨어분야에서 커플링이란 개념은 1974년 Stevens에 의해 처음 소개되었다. Stevens는 커플링을 "모듈사이의 연결관계에 의해 추출되는 모듈간의 연관간계의 강도(The measure of the strength of association established by a connection from one module to another)"로 정의하였다[8]. 따라서, 두 모듈간의 커플링 값이 높을수록 두 모듈간의 연관관계의 강도가 높기 때문에, 어느 한 모듈을 수정할 때 고려해야할 수정의 범위가 넓어지게 된다. 즉, 어떤 모듈이 다른 모듈과의 커플링이 높다면, 고려해야할 범위가 커지기 때문에, 모듈의 수정작업에 필요한 엔티티는 다른모듈과의 커플링이 낮은 모듈보다 더욱 많아지게 된다.

2.3 논리적커플링

논리적커플링은 두 엔티티 사이의 진화적 관점의 암시적 의존관계의 정도를 나타낸다[6]. 다시 말하면, 두 엔티티 사이의 논리적커플링은 소프트웨어가 진화하는 과정에서 두 엔티티가 얼마나 자주 짧은 범위의 시간안에 같이 변경되는지의 정도를 나타낸다[9]. 여기서 변경이란, 엔티티에 대해 버전컨트롤시스템의 Revision의 증가를 유발하는 행위로 정의하고, 두 개 이상의 Entity를 짧은범위의 시간안에 변경되는 이벤트를 Co-Change라고 한다. 따라서, 두 엔티티간의 논리적 커플링이 높다면, 이들은 과거에 Co-Change되는 빈도수가 높았고,

미래에 Co-Change될 확률이 높다고 말할 수 있다.

논리적커플링은 코드간의 정적 분석에 의해 측정될 수 있는 정적커플링과는 다른 측면의 커플링을 측정한다. 정적커플링과는 달리, 논리적커플링은 프로그램의 진화 관점에서 두 엔티티들 간의 커플링을 측정하는데 그 목적을 두고 있다.

3. 관련연구

본 장에서는 현재까지 연구되었던 논리적커플링 측정기법들에 대해서 살펴보고 각각의 장단점을 논의한다. 논리적커플링이란 개념은 1998년 Gall에 의해서 처음 제안되었다[2]. Gall의 논문에서는 20번의 릴리즈히스토리를 갖는 전화교환시스템(Telecommunication Switching System)에서 작성된 릴리즈히스토리 문서를 기반으로 시스템을 구성하는 모듈들 간의 커플링을 파악할 수 있는 기법을 제안하였다. Bevan은 릴리즈히스토리 문서가 추가적으로 요구되는 Gall의 기법의 한계점을 개선하기 위하여, 버전컨트롤시스템에 기본적으로 축적되는 변경 히스토리정보 만을 이용하여 논리적커플링을 추출하는 기법을 제안하였다[5]. 하지만, 이 기법은 측정된 논리적커플링값이 방향성을 갖지 못하고, 상대적인 중요도를 파악하기 힘든 단점을 갖는다. Zimmermann은 버전컨트롤시스템의 정보를 이용하여 메소드단위의 엔티티들간의 논리적커플링을 측정하는 기법을 연구하였다[6,10]. 이 논문에서 Zimmermann은 데이터마이닝 기법을 응용하여 관계의 방향성 및 관계의 상대적 중요도를 파악할 수 있는 기법을 제안하였다. Zimmermann은 이후 라인단위의 논리적커플링 측정기법을 제안하였으나[11], 이는 분석과정에 요구되는 시간복잡도가 상당히 크기 때문에 적용할 수 있는 범위가 제한되는 단점이 있다.

논리적커플링에 대한 기존의 관련연구들은 총 3가지의 문제점을 갖고 있는데, 각각의 문제점은 다음과 같다. 첫째로, 과거의 트랜잭션과 최근 트랜잭션의 중요도 차이를 고려하지 못하였다. 소프트웨어는 시간이 지남에 따라 다양한 관점에서의 진화가 이루어지는데, 이 중 아키텍처의 변경은 논리적커플링되는 엔티티들의 집합을 변화시킨다. 논리적커플링은 소프트웨어가 진화해감에따라 수반되는 아키텍처의 변경에 의해 변화될 수 있다. 아키텍처의 변경을 겪은 소프트웨어라면 아키텍처변경 이전에 발생되었던 트랜잭션보다 아키텍처변경 이후에 발생된 트랜잭션이 현재 소프트웨어의 논리적커플링을 더 잘 설명한다고 볼 수 있다. 따라서, 비교적 최근의 트랜잭션일수록 비교적 과거의 트랜잭션보다 논리적커플링의 결정에 더 많은 영향을 미친다고 할 수 있다.

둘째로, 개발자마다 서로 다른 변경사이클을 고려하지 못하였다. 오늘날 소프트웨어의 규모가 커짐에 따라, 단일 개발자가 개발한 소프트웨어는 찾아보기 힘들다. 많은 수의 소프트웨어가 두 명 이상의 개발자가 참여하여 개발되고, 이에 따라 버전컨트롤시스템에 쌓이는 히스토리정보의 특성은 참여하는 개발자의 수 만큼 다양하다. 이 중 개발자의 변경사이클의 길이는 논리적커플링을 추출해 내는데, 많은 영향을 주는 요소이다. 변경사이클의 길이는 개발자가 어느정도 규모의 트랜잭션을 유발시키는지와 직결된다. 변경사이클이 짧은 개발자는 소규모의 트랜잭션을 유발시킬 확률이 높고, 이들 트랜잭션은 하나의 유의미한 변경일 가능성이 높다. 반대로 변경사이클이 긴 개발자의 경우에는, 대규모의 트랜잭션을 유발시킬 확률이 높고, 따라서, 다수의 유의미한 변경의 집합이 트랜잭션에 포함될 가능성이 높다. 따라서, 변경사이클이 짧은 개발자일수록 유발시키는 트랜잭션의 규모도 작고, 그 트랜잭션이 논리적커플링에 미치는 영향도는 크다고 할 수 있다.

마지막으로, 트랜잭션을 유발하는 개발자의 숙련도를 고려하지 못하였다. 트랜잭션을 유발하는 개발자가 트랜잭션을 구성하는 엔티티들에 Ownership을 갖는지, 다시 말하면 숙련도가 높은지의 값은 측정되는 논리적커플링 값에 영향을 미친다고 할 수 있다.

4. 트랜잭션의 시점, 크기 및 개발자의 숙련도를 고려한 논리적커플링 측정기법

본 장에서는 다중관점 기반의 논리적커플링 측정기법에 대해서 기술한다. 먼저 4.1절에서는 본 논문에서 제안하는 특성기반 논리적커플링 추출기법의 개요에 대해서 논하고, 4.2절에서는 본 연구에서 제안하는 3가지 Factor들의 정의 및 이를 측정하는 방법을 설명한다. 마지막으로 4.3절에서는 Factor들이 할당된 트랜잭션에서 논리적커플링정보를 추출하는 방법을 기술한다.

4.1 기법 개요

그림 1은 본 논문에서 제공하는 기법의 전체적인 흐름을 나타낸다. 본 기법은 버전컨트롤시스템(Version Control System)에 기록되어 있는 변경 히스토리로부터 트랜잭션을 식별하는 단계부터 시작한다. 이 단계에서는 변경을 시간순서대로 추적하여 일정범위의 시간 안에서 이루어진 변경들을 하나의 트랜잭션으로 식별한다. 두 번째 단계에서는, 첫 번째 단계에서 식별된 트랜잭션들에 관점 별 중요도를 나타내는 트랜잭션시점, 트랜잭션크기, 그리고 숙련도 Factor들을 할당한다. 세 번째 단계에서는, 두 번째 단계에서 파악된 트랜잭션들의 관점 별 중요도를 기반으로 논리적커플링을 계산한다. 이 장의 나머지 절에서는 이 세 단계에 대해서 상세히 기술한다.

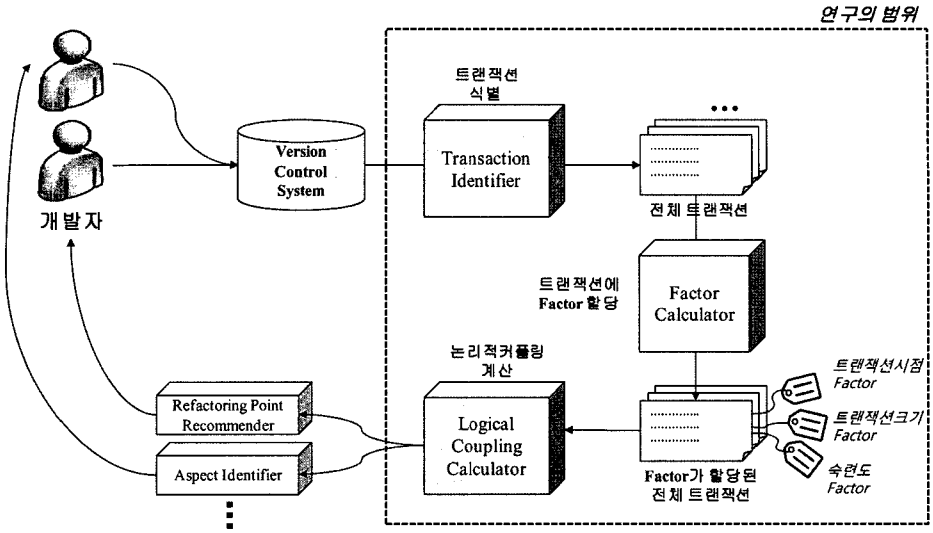

그림 1 제안하는 기법의 개요

4.2 1단계 : 트랜잭션 추출

본 연구에서 제안하는 기법의 첫 번째 단계는 버전컨트롤시스템으로부터 트랜잭션을 식별하는 것으로부터 시작한다. 버전컨트롤시스템에서 트랜잭션이란 일정범위의 시간 안에서 발생된 변경들의 집합을 말한다. 서브버전과 같이 자체적으로 트랜잭션을 지원하는 버전컨트롤시스템과는 달리 CVS같은 시스템에서는 명시적인 트랜잭션이 존재하지 않는다. 따라서, 이런 버전컨트롤 시스템에서 본 논문에서 제안하는 기법을 적용하기 위해서는 변경로그로부터 트랜잭션을 추출하는 단계가 필요하다.

그림 2는 트랜잭션을 식별하기 위해 Zimmermann이 제안한 슬라이딩 타임윈도우기법을 나타낸다[11]. 슬라이딩 타임 윈도우기법은 개발자별로 발생시킨 변경을 시간순서대로 따라가면서 트랜잭션을 식별한다. 이 기법의 시작은 특정개발자가 발생시킨 첫 번째 변경의 시점에 정해진 타임윈도우의 시작 지점을 일치시키는 데에서 출발한다(그림 2(a)). 이 때 타임윈도우 안에 개발자가 발생시킨 또 다른 변경이 존재한다면, 타임윈도우의 시작시점을 새롭게 식별된 변경의 시점으로 이동시킨다(그림 2(b)). 이와 같은 과정을 타임 윈도우 내에 새로운 변경이 발생되지 않을 때까지 반복하여, 그때까지 파악된 변경들의 집합을 트랜잭션으로 식별한다. 이 후 다음 변경의 시점에 타임윈도우의 시작점을 일치시켜 위 과정을 반복하여 트랜잭션을 식별한다(그림 2(c)).

4.3 2단계 : Factor 계산

이번 단계에서는 첫 번째 단계에서 식별된 트랜잭션들에 관점 별 중요도를 나타내는 트랜잭션시점, 트랜잭션크기, 그리고 숙련도 Factor들을 할당한다. 트랜잭션

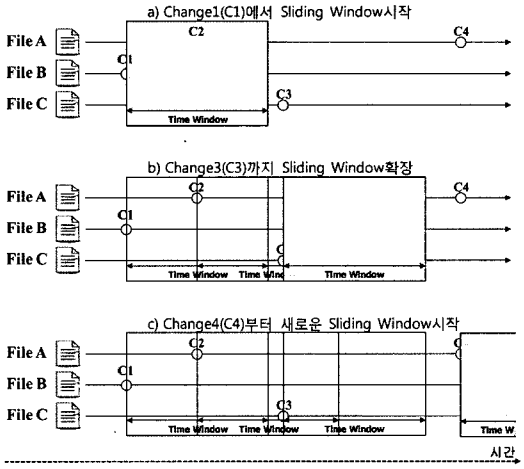

그림 2 슬라이딩 타임 윈도우기법에 의한 트랜잭션 식별

시점 Factor는 트랜잭션의 발생 시점에 따른 중요도를 반영하는 Factor로, 트랜잭션에 포함되는 엔티티들의 Revision이 최근일수록 더 높은 중요도 값을 갖는다. 트랜잭션크기 Factor는 트랜잭션의 크기에 따른 중요도를 반영하는 Factor로, 트랜잭션에 포함되는 엔티티들의 개수가 적을수록 더 높은 중요도 값을 갖는다. 마지막으로 숙련도 Factor는 변경된 엔티티들에 대해 얼마나 익숙한 개발자가 트랜잭션을 유발하였는지에 따른 중요도를 반영하는 Factor로 트랜잭션을 유발한 개발자가 트랜잭션에 포함되는 엔티티들에 대해 더 높은 Ownership을 가질수록 높은 중요도를 갖는다

4.3.1 트랜잭션시점 Factor

트랜잭션시점 Factor는 트랜잭션의 발생 시점에 따른 중요도를 반영하는 Factor로, 트랜잭션에 포함되는 엔티티들의 Revision이 최근일수록 더 높은 중요도 값을 갖는다. 버전컨트롤 시스템에 기록되어 있는 트랜잭션 Tx의 트랜잭션시점 Factor $F_{txTime}(Tx)$은 수식 (1)과 같이 구한다.

트랜잭션 Tx가 n개의 엔티티를 포함 하고
$x_i \in Tx$이고, $1 \le i \le n$이라고 할때

x_i 의 총 Revision 갯수는

$$\text{.....} TotalNumOf\,\text{Re}vision(x_i)$$

트랜잭션 Tx에서의 엔티티 x_i의 Index of Revision을

$$\text{.....} IndexOf\,\text{Re}vison(x_i, Tx)$$

트랜잭션 Tx에서의 엔티티 x_i 시점Factor를

$$\text{.....} TF(x_i, Tx) = \frac{IndexOf\,\text{Re}vison(x_i, Tx)}{TotalNumOf\,\text{Re}vision(x_i)}$$

라고 정의하면

Tx의 트랜잭션시점 Factor $T_{txTime}(Tx)$는 아래와 같이정의된다.

$$F_{txTime}(Tx) = \frac{\sum_{i=1}^{n} TF(x_i, Tx)}{n}$$

수식 (1) 트랜잭션시점 Factor의 계산식

4.3.2 트랜잭션크기 Factor

트랜잭션크기 Factor는 트랜잭션의 크기에 따른 중요도를 반영하는 Factor로, 트랜잭션에 포함되는 엔티티들의 개수가 적을수록 더 높은 중요도 값을 갖는다. 버전컨트롤 시스템에 기록되어 있는 트랜잭션 Tx의 트랜잭션크기 Factor $F_{txSize}(Tx)$는 수식 (2)와 같이 구한다.

트랜잭션 Tx가 n개의 엔티티를 포함 하고,
프로젝트 P의 최대트랜잭션크기를 ..... $MaxSizeOfTransaction(P)$
라고 정의하면

Tx의 트랜잭션크기Factor $F_{txSize}(Tx)$는 아래와 같이정의된다.
n = 1 일때,
$F_{txSize}(Tx) = 0$

n > 1 일때,
$F_{txSize}(Tx) = 1 - \log_{MaxSizeOfTransaction(P)-1} n - 1$

수식 (2) 트랜잭션크기 Factor의 계산식

4.3.3 숙련도 Factor

숙련도 Factor는 변경된 엔티티들에 대해 얼마나 익숙한 개발자가 트랜잭션을 유발하였는지에 따른 중요도를 반영하는 Factor로 트랜잭션을 유발한 개발자가 트랜잭션에 포함되는 엔티티들에 대해 더 높은 Ownership을 가질수록 높은 중요도를 갖는다. 숙련도 Factor의 계산식은 프로그램 엔티티와 그렇지 않은 엔티티가 다르다. 먼저 프로그램 엔티티가 아닌 엔티티들로 구성된 트랜잭션의 숙련도는 수식 (3)과 같이 구한다.

트랜잭션 Tx가 n개의엔티티를 포함 하고, 개발자 A가 Tx를 발생시켰을때
$x_i \in Tx$이고, $1 \le i \le n$이라고 할때

x_i 의총 Revision 갯수는

$$\text{.....} TotalNumOf\,\text{Re}vision(x_i)$$

개발자 A가 발생시킨 x_i의 Revision 갯수는

$$\text{.....} NumOf\,\text{Re}vision(x_i, A)$$

트랜잭션 Tx에서의 리소스 x_i의 숙련도 Factor를

$$\text{.....} EF(x_i, Tx) = \frac{NumOf\,\text{Re}vision(x_i, A)}{TotalNumOf\,\text{Re}vision(x_i)}$$

라고 정의하면

Tx의 숙련도 Factor $F_{expertise}(Tx)$는 아래와 같이정의된다.

$$F_{expertise}(Tx) = \frac{\sum_{i=1}^{n} EF(x_i, Tx)}{n}$$

수식 (3) Non-Program 엔티티의 숙련도 Factor 계산식

다음으로 프로그램 엔티티로 구성된 트랜잭션의 숙련도는 수식 (4)와 같이 구한다.

트랜잭션 Tx가 n개의엔티티를 포함 하고, 개발자 A가 Tx를 발생시켰을때
$x_i \in Tx$이고, $1 \le i \le n$이라고 할때

x_i 의 추가된라인수 누적

$$\text{.....} AccumulationOfTotalChangedLines(x_i)$$

A가 추가한 x_i의라인수 누적

$$\text{.....} AccumulationOfChangedLines(x_i, A)$$

트랜잭션 Tx에서의리소스 x_i 숙련도 Factor를 .. $EF(x_i, Tx)$

$$= \frac{AccumulationOfChangedLines(x_i, A)}{AccumulationOfTotalChangedLines(x_i)}$$

라고 정의하면

Tx의숙련도 Factor $F_{expertise}(Tx)$는 아래와 같이정의된다.

$$F_{expertise}(Tx) = \frac{\sum_{i=1}^{n} EF(x_i, Tx)}{n}$$

수식 (4) Program 엔티티의 숙련도 Factor 계산식

4.4 3단계 : 논리적커플링 계산

이번 단계에서는 두 번째 단계에서 파악된 트랜잭션들의 관점 별 중요도를 기반으로 엔티티들 간의 논리적 커플링을 계산한다. 그림 3은 $allT$를 전체 트랜잭션 집

```
Input :
 e  // 논리적커플링을 판단하려는 엔티티
 allT  // Factor들이 할당된 트랜잭션 집합
 minimumS  // 최소 지지도(Support)
 minimumC  // 최소 신뢰도(Confidence)

Output :
 C  // 논리적커플링 예측 집합

Algorithm :
C = Ø ;      // r과 Logically Coupling된 엔티티들의 집합
 relatedT = Ø ; // e이 포함되어 있는 트랜잭션 중 모든 중요도 값이
        // 과락을 넘은 트랜잭션 집합
relatedE = Ø ; // relatedT에 포함되어 있는 e을 제외한 엔티티들의 집합
sumOfFactors = 0;

// 1단계
for each Ti ∈ allT do  // 모든 트랜잭션 중에서
  if e ∈ Ti and          // e이 포함되어 있고
   Tr->txTimeFactor >= 0.1 and // 트랜잭션시점 Factor가 0.1보다 크고
   Tr->txSizeFactor >= 0.1 and // 트랜잭션크기 Factor가 0.1보다 크고
   Tr->expertiseFactor >= 0.1 then // 숙련도 Factor가 0.1보다 크다면
    relatedT = relatedT ∪ Ti;
     sumOfFactors = Tr->txTimeFactor + Tr->txSizeFactor +
                    Tr->expertiseFactor ;


// Step 2
for each Ti ∈ relatedT do
 for each Rj ∈ Ti do
  if Rj ≠ r then
     relatedE = relatedE ∪ Rj;

// Step 3
for each Ei ∈ relatedE do
 for each Tj ∈ relatedT do
  if Ei ∈ Tj then
   Er->support = Er->support + 1 ;
      Er->sumOfFactors = Tr->txTimeFactor + Tj->txSizeFactor +
                         Tj->expertiseFactor ;
    Er->confidence = Er->sumOfFactors / sumOfFactors ;
    If Er->support >= minimumS and Er->confidence >= minimum then
  C = C ∪ Ei;
```

그림 3 다중관점기반의 논리적커플링 추출 알고리즘

합으로 갖는 소프트웨어의 히스토리정보로부터 엔티티 e에 논리적커플링 되어 있는 엔티티들을 찾아내는 알고리즘을 기술한다.

그림 3에 기술된 알고리즘을 설명하기 위해 가상의 프로젝트로부터 추출된 트랜잭션과 여기에 할당된 3가지 Factor들의 값과 그 합을 표 1에 나타내었다. 표 1에 기술된 트랜잭션 히스토리를 기반으로 e1에 논리적커플링된 엔티티들을 찾아내는 과정은 다음과 같다.

- **Step 1** : e1과 논리적커플링되어 있는 엔티티들을 찾아내기 위한 가장 처음 단계로, e1이 포함되어 있고, 모든 Factor의 값들이 0.1이상인 트랜잭션들을 선정하고, 선정된 트랜잭션의 Factor들의 합을 구한다.
  - 선정된 트랜잭션 : T100, T99, T98
  - Factor들의 합 : 6.62
- **Step 2** : Step1에서 선정된 트랜잭션(T100, T99, T98)에 속해 있는 엔티티들 중 e1을 제외한 엔티티들 추출.
  - 추출된 엔티티 : e2, e3, e4, e5, e6, e7
- **Step 3** : Step2에서 추출된 각각의 엔티티들이 속해 있는 트랜잭션 개수를 Support값으로 구하고, e1이 등장한 전체 트랜잭션의 Factor들의 합과 e1과 각각의 엔티티들이 동시에 출몰한 트랜잭션의 Factor들의 합과의 비율을 Confidence값으로 구한다.
  - e2 - support = 2, confidence = (2.5 + 1.49) / 6.62 = 0.602
    e3 - support = 2, confidence = (2.5 + 2.63) / 6.62 = 0.774
    e4 - support = 1, confidence = (1.49) / 6.62 = 0.225
    e5 - support = 1, confidence = (1.49) / 6.62 = 0.225
    e6 - support = 1, confidence = (1.49) / 6.62 = 0.225
    e7 - support = 1, confidence = (2.63) / 6.62 = 0.397

그림 4는 최소 Support와 Confidence값이 각각 2와 0.7인 경우와, 1과 0.3인 경우에 추천되는 논리적커플링 엔티티의 변화를 나타낸다. 최소 Support와 Confidence 값이 낮을수록 더 많은 엔티티들이 추천되지만, 논리적 커플링의 강도가 낮은 엔티티들이 추천된다.

5. 기법 검증

본 논문에서 제안한 논리적커플링측정기법의 효율을 검증하기 위해서 CVS저장소가 열려있는 5개의 오픈소스 프로젝트를 검증대상 프로젝트로 선정하였다. 표 2는 대상으로 선정된 프로젝트를 기술하고 있다.

선정된 프로젝트는 각각 14명에서 141명의 개발자가 참여하여 개발된 프로젝트로써, 개발자의 다양성이 충분히 반영되어 있다고 할 수 있다. 그리고, 검증 대상 프

표 1 Factor들이 할당된 가상의 트랜잭션 예제

	트랜잭션시점 Factor	트랜잭션크기 Factor	숙련도 Factor	트랜잭션 중요도
T100(e1,e2,e3)	1	0.85	0.65	2.5
T99(e1,e2,e4,e5,e6)	0.99	0.30	0.2	1.49
T98(e1,e3,e7)	0.98	0.85	0.8	2.63
…	…	…	…	…
T70(e1,e2,e5,e10…)	0.7	0.05	0.78	1.53
…	…	…	…	..
T2(e1,e4)	0.02	1	1	2.02
T1(e4,e7,e9)	0.01	0.85	1	1.86

리소스	Support	Confidence
r3	2	0.774
r2	2	0.602
r7	1	0.397
r5	1	0.225
r6	1	0.225
r7	1	0.225

최소 Support = 2, 최소 Confidence = 0.7

최소 Support값이 높을 수록 정확하지만, 결정되는 리소스 개수는 작다.

최소 Support = 1, 최소 Confidence = 0.3

최소 Support값이 낮을 수록 결정되는 리소스 개수는 많지만, 정확도는 떨어진다.

그림 4 최소 Support, Confidence에 따른 논리적커플링결과 변화

표 2 검증에 사용된 2개의 오픈소스 프로젝트

프로젝트 명	도메인	개발자 수	개발기간	트랜잭션 수
Azureus2	P2P 클라이언트	34 명	2003.7~ 2009.6.	15,163
Jboss	J2EE 컨테이너	141 명	2000.4~ 2006.6	9,160
Eclipse Compare Plug-in	개발환경	14 명	2001.5~ 2009.6	1,246
Pgsql	DBMS	31 명	1996.7~ 2009.5	31,666
Php3	웹 프레임워크	72 명	1997.8~ 2002.6	6,681

로젝트 모두 5년 이상의 수명을 갖고 있고 1000개 이상 3만여 개 미만의 트랜잭션이 히스토리정보로 축적되어 있다. 따라서 충분히 많은 히스토리정보가 기록되었다고 할 수 있다.

5.1 정확도 측정방법

수식 (5)는 예측정확도를 산출하기 위한 식을 나타낸다. 어떤 프로젝트에 대해 논리적커플링의 예측정확도는 3번과 4번식에 의해 표현된다. Precision값은 어떤 엔티티가 주어졌을 때 해당 엔티티와 실제로 논리적커플링 되어있는 엔티티들을 얼마나 많이 찾는지의 정도를 나타낸다. Recall값은 주어진 엔티티와 논리적커플링 되어 있다고 예측된 엔티티들이 정말로 얼마나 정확히 커플링이 되어있는지의 정도를 나타낸다. 주어진 트랜잭션집합을 기반으로 더 높은 Precision과 Recall값을 집합을 예측해 낼 수록 더 나은 방법이라 할 수 있다.

엔티티 $e \in Tx$에대하여,

$A_e = Tx - \{e\}$

$E_e = \{x \mid x$는 e와 로지컬커플링되었을것으로예측된엔티티$\}$라고 할때

e에의해예측된 로지컬커플링의 Precision과 Recall은 각각

$$P(e) = \frac{|A_e \cap E_e|}{|A_e|} \quad \text{----------(1)}$$

$$R(e) = \frac{|A_e \cap E_e|}{|E_e|} \quad \text{----------(2)}$$

그리고, Tx에대한예측정확도인 Precision과 Recall은 각각

$$P(Tx) = \frac{\sum_{e \in Tx} P(e)}{|Tx|} \quad \text{----------(3)}$$

$$R(Tx) = \frac{\sum_{e \in Tx} R(e)}{|Tx|} \quad \text{----------(4)}$$

수식 (5) 예측 정확도 산출

5.2 정확도 기여 검증

실험을 위해 본 논문에서는 검증대상 프로젝트에 대해서 가장 최근에 발생된 크기가 2이상 10미만인 트랜잭션 50개(Eclipse Compare Plug-in의 경우 가장 최근의 20개 트랜잭션을 선정)를 논리적커플링 된 엔티티집합으로 선정하였다. 선정된 각각의 트랜잭션에 대하여 트랜잭션에 포함된 어떤 엔티티가 주어졌을 때 트랜잭션의 나머지 엔티티들을 얼마나 정확히 예측할 수 있는지를 (3)과 (4)식을 이용하여 Precision과 Recall 형태로 측정하였고, 이 값을 모든 트랜잭션에 대해 측정하여 평균을 내었다. 이에 추가적으로, 기법의 실용성을 파악하기 위해 특정 엔티티가 주어졌을 때 논리적커플링 된 엔티티들이 추천되는 비율인 추천률을 측정하였다. 이와 같은 과정을 논문에서 제안된 기법과, 현재 가장 정확도가 높은 Zimmermann이 제안한 기법을 같이 수행하여 그 결과값을 비교해 보았다.

표 3은 최소 Support값을 3으로 하고 최소 Confidence값을 0.0에서 0.8까지 0.1단위로 증가시켜가며 측정한 두 기법의 Precision과 Recall값, 그리고 추천률에 대한 평균값의 차이를 나타낸다. 검증의 대상으로 선정된 5개의 프로젝트 모두 Precision값이 적게는 0.61%에서 많게는 9.78% 개선되었고, Recall값은 적게는 0.34%에서 많게는 22.02% 개선되었다. 추천률은 4개의 프로젝트에 대해서 감소되었지만, 감소된 비율은 Precision과 Recall의 개선으로 극복될 수 있는 것으로 보인다.

5.3 Azureus2 프로젝트에서의 검증결과

이번 섹션에서는 앞에서 기술한 검증 방법을 Azureus2프로젝트에 적용한 결과를 기술한다(이번 섹션에

표 3 최소 Support값이 3일 때 두 기법의 정확도 차이

	Precision 차이	Recall 차이	추천률
Azuresu2	+9.78%	+22.02%	-0.96%
JBoss	+5.83%	+16.43%	-5.75%
Eclipse Compare Plug-in	+0.61%	+7.40%	-1.07%
Pgsql	+1.78%	+7.58%	-4.29%
Php3	+1.70%	+0.34%	+3.67

서 기술하는 논리적커플링은 모두 메소드 및 필드 단위 엔티티간의 논리적커플링이다). Azureus2 프로젝트는 P2P의 일종인 Bittorrent의 Java버젼 오픈소스 클라이언트이다[12]. 표 4는 검증을 위한 실험의 전체 결과를 보여준다. 전체적으로 평균값은 본 논문에서 제안한 방법으로 측정한 결과가 Zimmermann이 제안한 방법으로 측정한 결과값보다 더 나은 결과를 보여주고 있다. 그리고 Support값이 높을수록 더 높은 Precision과 Recall 값을 보여주고 있다.

표 4 Azuresu2 프로젝트의 검증결과 정리

	최소 Support = 1		최소 Support = 3		최소 Support = 5	
	평균 Precision	평균 Recall	평균 Precision	평균 Recall	평균 Precision	평균 Recall
My Approach	24.46%	18.16%	52.52%	70.22%	80.74%	90.77%
Zimmermann Approach	22.55%	16.33%	42.74%	48.19%	70.69%	81.70%
차이	1.91%	1.83%	9.78%	22.02%	10.05%	9.07%

5.4 JBoss 프로젝트에서의 검증결과

이번 섹션에서는 동일한 방법을 JBoss프로젝트에 적용한 결과를 기술한다(이번 섹션에서 기술하는 논리적커플링은 모두 메소드 및 필드 단위 엔티티간의 논리적커플링이다). JBoss는 J2EE 컨테이너를 Java언어로 구현한 오픈소스 프로젝트이다[13]. 표 5는 실험결과를 나타낸다. Azureus2의 실험결과와 마찬가지로 전체적인 Precision과 Recall값의 평균값은 본 논문에서 제안한 방법으로 측정한 결과가 Zimmermann이 제안한 방법으로 측정한 결과값보다 더 나은 결과를 보여주고 있다.

표 5 JBoss프로젝트의 검증결과 정리

	최소 Support = 1		최소 Support = 3		최소 Support = 5	
	평균 Precision	평균 Recall	평균 Precision	평균 Recall	평균 Precision	평균 Recall
My Approach	33.99%	38.39%	33.02%	52.71%	23.18%	61.62%
Zimmermann Approach	28.66%	31.34%	27.18%	36.28%	23.02%	40.24%
차이	5.34%	7.04%	5.83%	16.43%	0.16%	21.38%

JBoss의 Precision과 Recall값은 두 방법 모두 Azureus2의 실험결과보다 현저히 낮게 측정되었는데, 이는 JBoss 프로젝트의 다음과 같은 특성 때문이라고 보여진다.

- 축적된 히스토리정보가 트랜잭션 개수가 30%가량 적다.
- Sliding time window으로 트랜잭션추출이 부정확함. 정해진 Time window를 벗어난 커밋이 종종 발생함.

5.5 Pgsql 프로젝트에서의 검증결과

이번 섹션에서는 마지막으로 Pgsql프로젝트에 적용한 결과를 기술한다(이번 섹션에서 기술하는 논리적커플링은 모두 파일 단위 엔티티간의 논리적커플링이다). Pgsql은 C언어로 구현된 경량의 오픈소스 DBMS(Database Management System)이다[14]. 표 6는 전체 실험결과를 보여주고 있는데, 전체적으로 Precision의 평균값은 두 방법간의 차이가 없어 보이지만, Recall값은 본 논문에서 제안한 방법으로 측정한 결과가 Zimmermann이 제안한 방법으로 측정한 결과값보다 더 나은 결과를 보여주고 있다.

표 6 Pgsql프로젝트의 검증결과 정리

	최소 Support = 1		최소 Support = 3		최소 Support = 5	
	평균 Precision	평균 Recall	평균 Precision	평균 Recall	평균 Precision	평균 Recall
My Approach	37.79%	40.19%	38.72%	48.60%	36.83%	49.95%
Zimmermann Approach	37.82%	38.13%	36.94%	41.01%	36.32%	41.34%
차이	-0.03%	2.06%	1.78%	7.58%	0.51%	8.61%

6. 지원도구 - MDT(Multi-Dimensional Logical Coupling Measurement Tool)

본 장에서는 4장에서 제안한 다중관점 기반의 논리적 커플링 측정기법을 지원하기 위해 구현된 MDT(Multi-Dimensional Logical Coupling Measurement Tool)를 설명한다.

6.1 도구의 아키텍처

그림 5는 MDT의 아키텍처를 나타낸다. MDT는 논문에서 제안한 3단계의 절차를 구현한 3개의 컴포넌트(Transaction Identifier, Factor Attach, Logical Coupling Calculator Component)와 대표적인 버전컨트롤시스템인 CVS에 접근 서비스를 제공하는 CVS Repository Adapter, 그리고 중간데이터를 DBMS(Data Base Management System)에 기록하고, 조회하는 서비스를 제공하는 Repository Adapter로 구성된다. MDT도구는 개발 플랫폼으로 광범위하게 사용되고 있는 Eclipse

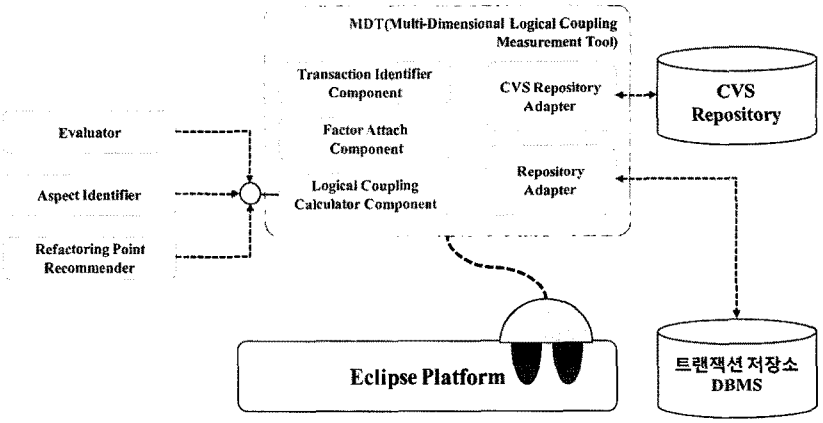

그림 5 MDT 도구의 아키텍처

Platform위에 Plug-in형태로 개발되었다. 도구는 기본적으로 트랜잭션 식별과 Factor할당 및 논리적커플링값 계산기능을 제공한다. 이 중 트랜잭션 식별과 Factor할당 기능은 이클립스의 메뉴형태로 구현되었고, 논리적커플링값 계산기능은 프로그램인터페이스형태로 구현되었다.

MDT의 Logical Coupling Calculator Component는 특정 엔티티를 입력으로 받아서, 입력된 엔티티에 대해 논리적커플링 되어있는 엔티티들을 Support와 Confidence로 표현되는 커플링정도의 정보를 포함하여 리턴하는 인터페이스를 제공한다. 이 인터페이스는 향후, 논리적커플링을 응용한 연구의 시작점이 될 것이다.

7. 결론 및 향후 연구

효과적인 유지보수방안 마련은 소프트웨어의 수명이 점점 길어짐에 따라서 증가하고 있는 유지보수비용을 줄이기 위해 필수적이다. 효과적인 유지보수방안 마련을 위한 한 방법은 유지보수비용을 비용대비 효과가 높은 곳에 집중하는 것이다. 논리적커플링은 소프트웨어의 진화관점에서 유지보수 비용대비 효과가 높은 곳을 찾아내기 위한 정량적 데이터를 제공한다.

본 연구는 세가지 관점의 중요도를 나타내는 트랜잭션시점, 크기 및 숙련도 Factor를 정의하고, 이 중요도 값을 고려하여 논리적커플링을 측정하는 기법 제안하였다. 제안된 측정기법을 지원하기 위하여 Eclipse의 Plug-in형태의 도구를 개발하였고, 이 도구는 측정된 논리적커플링값을 외부로 서비스하는 인터페이스를 제공하여 논리적커플링값을 기반으로 한 응용연구에 기여할 것으로 보인다.

본 연구에서 제안한 논리적커플링 측정기법의 효율성을 검증하기 위하여, 서로 다른 도메인을 갖는 5개의 오픈소스 프로젝트를 선정하여 현재 가장 높은 정확도를 갖는 Zimmermann이 제안한 기법[6]과의 정확도를 비교하였다. 검증결과, 낮은 Support값 조건에서는 큰 차이를 보이지 않았으나, 높은 Support값에서는 전체 Confidence값에 대하여 높은 정확도를 보여주었다.

향후 연구에서는 본 연구에서 제안한 세가지 관점의 중요도값이 논리적커플링에 미치는 영향도를 좀더 상세히 연구할 필요가 있다. 이 중요도 값은 소프트웨어에 참여한 개발자의 특성 및 프로젝트의 정책에 따라 논리적커플링에 미치는 상대적 영향도가 다를 것으로 예상되기 때문에, 영향도를 파악한다면 좀 더 향상된 정확도를 갖는 측정기법이 기대된다. 추가적으로, 더 풍부한 정보를 기반으로 하여 숙련도 Factor를 측정함으로써, 한계점을 갖는 숙련도 Factor를 개선할 필요가 있다. 마지막으로, 제안된 세가지 관점의 중요도를 활용하여 새로운 논리적커플링 응용방안을 마련할 예정이다.

참 고 문 헌

[1] L. C. Briand, J. W. Daly, and J. K. W¨ust, "A Unified Framework for Coupling Measurement in Object-Oriented Systems," *IEEE Transactions on Software Engineering*, 25(1), pp.91-121, 1999.

[2] Harald Gall, Karin Hajek, Mehdi Jazayeri, "Detection of Logical Coupling Based on Product Release History," *International Conference on Software Maintenance 1998*, pp.190-197, 1998.

[3] Erik Arisholm, Lionel C. Briand, Audun Føyen, "Dynamic Coupling Measurement for Object-Oriented Software," *IEEE Transaction on Software Engineering*, 30(8), pp.491-506, 2004.

[4] Denys Poshyvanyk, Andrian Marcus, "The Conceptual Coupling Metrics for Object-Oriented Systems," *International Conference on Software Maintenance 2006*, pp.469-478, 2006.

[5] Jennifer Bevan, E. James Whitehead, Jr., "Identi-

fication of Software Instabilities," *Working Conference on Reverse Engineering 2003*, pp.134-145, 2003.
[6] Thomas Zimmermann, Stephan Diehl, Andreas Zeller, "How history justifies system architecture (or Not)," *International Workshop on Principles of Software Evolution 2003*, pp.73-83, 2003.
[7] Ian Sommerville, "Software Engineering 8th Edition," Addison Wesley, 2007.
[8] W. Stevens, G. Myers, and L. Constantine, "Structured Design," *IBM Systems J.*, 13(2), pp.115-139, 1974.
[9] Marco D'Ambros, Michele Lanza, "Reverse Engineering with Logical Coupling," *13th Working Conference on Reverse Engineering*, pp.189-198, 2006.
[10] Thomas Zimmermann, Peter Weißgerber, Stephan Diehl, Andreas Zeller, "Mining Version Histories to Guide Software Changes," *IEEE Transactions on Software Engineering*, 31(6), pp.429-445, 2005.
[11] Thomas Zimmermann, Sunghun Kim, Andreas Zeller, E. James Whitehead, Jr., "Mining version archives for co-changed lines," *Mining Software Repositories 2006*, pp.72-75, 2006.
[12] http://azureus.sourceforge.net/
[13] http://www.jboss.org/
[14] http://www.pgsql.com/

심 빈 구
2001년 중앙대학교 컴퓨터공학과 졸업(학사). 2001년~2003년 (주)JKDSoft 아키텍트. 2004년~2007년 (주) 소프트웨어 크래프트, 선임 컨설턴트. 2007년~2009년 서강대학교 컴퓨터학과 졸업(석사) 2009년~현재 (주) 드리머 Product팀 팀장. 관심분야는 Empirical software engineering, Software repository mining, 아키텍처

김 진 태
1998년 서강대학교 컴퓨터공학 학사. 2000년 서강대학교 컴퓨터공학 석사. 2001년 데이콤시스템 테크놀로지 소프트웨어 엔지니어. 2005년 서강대학교 컴퓨터공학 박사. 2009년 삼성전자 DMC 연구소 책임연구원. 2009년 성균관대학교 정보통신공학과 겸임교수. 현재 서강대학교 컴퓨터공학과 연구교수. 관심분야는 Empirical software engineering, Software reengineering, Requirements engineering

박 수 용
1986년 서강대학교 전자계산학(공학사) 1988년 Florida State University 전산학(석사). 1995년 George Mason University 정보기술학(박사). 1996년~1998년 TRW Senior Software Engineer. 1998년~현재 서강대학교 컴퓨터학과 정교수 관심분야는 Requirements engineering, Robot Software, Self healing SW