프로덕트라인 아키텍처의 실용적 설계기법
(A Practical Techniquefor Designing Product Line Architecture)

장 수 호 † 라 현 정 †† 김 수 동 †††
(Soo Ho Chang) (Hyun Jung La) (Soo Dong Kim)

요 약 프로덕트라인 공학은 여러 어플리케이션들이 공유할 수 있는 핵심자산을 사용하는 대표적인 소프트웨어 재사용 방법으로 넓게 수용되고 있다. 프로덕트라인 공학의 핵심자산의 중요한 구성요소로 프로덕트라인 아키텍처(Product Line Architecture, PLA)가 있다. 그러나 PLA에 대한 대부분의 연구는 재사용 자산으로서 공통성 및 가변성(C&V)에 대한 표현 및 설계에 대한 상세한 지침이 미비하다. 본 논문에서는 PLA 의 설계 프로세스와 상세 수준의 지침을 제안한다. 특히 PLA 가변성의 체계적인 정의를 위해 가변점의 종류에 대한 PLA 의사결정 모델(Decision Model)을 제시한다. 제안된 프로세스와 PLA가변성 정의로 프로덕트라인 공학의 효율적인 실무적 접근을 예상할 수 있다.

키워드 : 프로덕트라인 공학, 아키텍처, 프로세스, 가변성

Abstract Product Line Engineering (PLE) has been widely accepted as a representative software reuse methodology by using core assets. Product line architecture (PLA) is a key element of core assets. However, current research works on designing PLA do not provide sufficient and detailed guidelines of defining PLA and reflecting variability in the architecture. In this paper, we present a reference model of PLA and propose a process to design PLA with detailed instructions. Especially architectural variability is codified by describing decision model depending variation points and traced through PLA activities. The proposed process would make it feasible to apply PLE to practice areas.

Key words : Product Line Engineering, Architecture, Process, Variability

1. 서 론

프로덕트라인 공학은 여러 어플리케이션들이 공유할 수 있는 핵심자산을 사용하는 대표적인 소프트웨어 재사용 방법으로 넓게 수용되고 있다. 핵심 자산은 제품 계열에 속하는 패밀리 멤버들이 어플리케이션을 만드는데 기초가 되는 모든 자산을 포함하며, PLA, 컴포넌트, 결정 모델이 포함된다[1,2]. PLA는 패밀리 멤버들이 공통적으로 사용할 수 있는 아키텍처로, 제품 계열에 속하는 제품들의 구조를 정의하고 컴포넌트의 인터페이스 명세를 제공하여 컴포넌트만큼 중요한 재사용 단위이다. 그러므로 PLA를 중심으로 핵심 자산을 설계하면 더욱 효과적이다[3]. 그러나 일반 소프트웨어 아키텍처와 PLA의 가장 큰 차이점인 가변성을 체계적으로 다루지 못하여 공통성 및 가변성에 대한 표현 방법이 구체적으로 제시되지 않고, 아키텍처 드라이버, 뷰, 스타일과 같은 아키텍처 설계에 중요한 역할을 하는 요소뿐만 아니라 아키텍처 가변성이 PLA에 적용되는 프로세스와 상세한 지침이 미비하다.

본 논문에서는 PLA에 대한 기존 연구의 한계점을 극복하기 위해 2장에서 제품 계열 공학과 관련된 문헌에서 제안된 아키텍처와 이를 바탕으로 PLA의 구성요소에 대해 알아보고, 3장에서 PLA의 설계 프로세스를 제안하고, 4장에서는 프로세스의 각 단계의 활동 및 상세 지침과 PLA 의사 결정 모델을 제안한다. 그리고, 5장에서는 사례 연구를 통해 본 논문에서 제시한 프로세스를 평가하며 6장에서 결론을 맺는다. 제안된 프로체스와 PLA가변성 정의로 프로덕트라인 공학의 용이한 실무적 접근을 예상할 수 있다.

2. 기반연구

2.1 기존의 PLA 프로세스 연구

FORM은 휘쳐 단위로 재사용 가능한 아키텍처와 컴

· 이 논문은 2004년도 한국학술진흥재단의 지원에 의하여 연구되었음 (KRF-2004-005-D00172)
† 학생회원 : 숭실대학교 컴퓨터학과
shchang@otlab.ssu.ac.kr
†† 학생회원 : 숭실대학교 컴퓨터학과
hjla@otlab.ssu.ac.kr
††† 종신회원 : 숭실대학교 컴퓨터학과 교수
sdkim@comp.ssu.ac.kr
논문접수 : 2004년 11월 25일
심사완료 : 2005년 2월 1일

포넌트를 개발하고, 이로부터 한 어플리케이션을 유도하는 방법을 제공한다[4,5]. 이 방법론은 도메인 공학과 어플리케이션 공학으로 구성된다. 도메인 공학에서는 휘쳐 모델은 능력(Capability), 응용 환경(Operating Environment), 도메인 기술(Domain Technologies), 구현 기술(Implementation Techniques)의 계층 구조로 조직화되고, 휘쳐 모델과 제품 계열 요구사항을 기반으로 개념적 아키텍쳐를 설계한다. 이런 개념적 아키텍쳐는 각각의 특징에 따라 프로세스 아키텍쳐와 배치 아키텍쳐로 정제된다. FORM은 가변성을 선택(Optional), 대안(Alternative), 필수(Mandatory)로 분류하였다. 휘쳐 모델을 아키텍쳐로 표현하기 위해 휘쳐 모델의 네 계층에 있는 요소들은 아키텍쳐에 반영되고 설계되며 정제되는 과정을 거친다.

Bosch는 시스템 패밀리 소프트웨어 아키텍쳐를 설계하는 방법을 제안하였다[1]. 이 방법은 비즈니스 케이스 분석, 스코핑, 시스템과 휘쳐 계획 단계, 시스템 패밀리 아키텍쳐 설계, 컴포넌트 요구사항 명세, 검증의 6단계로 이루어진다. 패밀리 아키텍쳐는 아키텍쳐 설계에 중요한 역할을 하는 기능적 휘쳐인 아크타입(Archetype)를 기반으로 설계되고, 품질 요구사항에 대해 평가되며, 아키텍쳐의 품질 요구사항을 향상시키기 위해 품질 요구사항을 기능성으로 변환하는 단계를 거쳐 설계된다. 아키텍쳐 스타일, 아키텍쳐 패턴, 디자인 패턴을 적용하는 방법 외에, 패밀리 아키텍쳐는 가변적 요구사항과 아키텍쳐의 선택적인 부분을 반영하고, 컴포넌트 간의 충돌을 제거함으로써 품질 요구사항을 아키텍쳐의 기능으로 변환시킨다.

QADA는 품질 요소기반으로 아키텍쳐를 분석 설계하는 방법으로[6], 요구사항 분석, 개념적 아키텍쳐 설계와 분석, 구체적 아키텍쳐 설계와 분석 단계로 구성된다. 요구사항 분석 단계에 시스템 컨텍스트와 기술적 속성을 분석하고, 개념적 아키텍쳐 설계와 분석 단계에서는 추상적인 레벨에서 아키텍쳐를 설계하고, 아키텍쳐 스타일과 뷰가 품질요소를 잘 반영하는지 분석하고 평가한다. 그리고, 구체적 아키텍쳐 설계와 분석 단계에서는 개념적 아키텍쳐 명세를 이용하여 좀더 상세한 레벨에서 아키텍쳐를 설계하고, 품질 요소에 대해 분석 평가한다. 이 방법론은 아키텍쳐를 구조적 뷰, 행위적 뷰, 배치 뷰 세 가지 관점을 이용하여 표현한다. 가변성 분석과 표현은 개념적 아키텍쳐 분석 단계에서 이루어지고, 가변점 명세와 제품 계열 패턴을 이용하여 가변성이 설계된다.

Ceron은 참조 아키텍쳐를 개발하는 프로세스를 제안하였다[7]. 참조 아키텍쳐는 먼저 제품 계열 요구사항을 이용하여 소코핑, 아키텍쳐 스타일 선택함으로써 공통 요구사항만 포함하는 아키텍쳐를 설계한다. 그리고, 제품에 특정한 요구사항을 기반으로 아키텍쳐에 가변성 적용하는 단계를 거쳐 공통성과 가변성을 모두 포함한 완전한 참조 아키텍쳐가 개발된다. 이 연구는 기능적 휘쳐와 비기능적 휘쳐를 아키텍쳐에 적용함으로써 참조 아키텍쳐와 가변성 충돌 문제를 지적하였고, 참조 아키텍쳐는 공통 요구사항과 아키텍쳐 스타일이 변하지 않는 범위 내에 진화해야 함을 제시한다.

Thiel은 QUASAR이라고 불리는 고품질의 PLA를 설계하는 프레임워크를 제안하였다[8,9]. QUASAR은 준비, 설계, 평가 세 단계로 구성된다. 준비 단계는 아키텍쳐 설계 초기 단계에 고려되어야 할 사항을 지원하고, 설계 단계에서는 아키텍쳐 뷰와 각 뷰마다 존재하는 가변성을 설계하며, 평가 단계에서는 아키텍쳐가 품질 요구사항을 만족하는지에 대해 분석한다. 효과적으로 PLA과 가변성을 통합하기 위해, 각 아키텍쳐뷰에서 발생하는 가변점, 가변점을 인스턴시에이션(Instantiation)방법과 해결방법(Resolution Rule)에 대해 가변성을 문서화하는 가이드라인을 제시하였다.

이 관련 연구들은 PLA에 포함되는 요소들을 암시적으로 정의하였고, PLA를 설계하는 개략적인 프로세스를 제안하였다. 그리고, PLA를 설계하는데 가변성 표현과 문서화 필요성에 대해 언급하였다. 그러나, PLA의 구성요소들이 프로세스에 적용되는 지침과 산출물 양식 등이 구체적으로 제시되지 않았다. 특히, 아키텍쳐에 가변성이 있음은 모든 연구에서 제시하였으나, 구성 요소에 따른 상세한 수준에서의 가변성 형태 및 표현 방식은 제시되지 않았다.

2.2 PLA의 구성요소

PLA에 대한 다른 연구에서의 아키텍쳐 구성요소를 정리하면 다음과 같다. PuLSE 방법론[10]에서는 구체적으로 컴포넌트와 관계를 명시하고 있지 않지만 명세(모델)에 그 내용이 내포되어 있고, 다른 방법론과 달리 가변성 모델을 아키텍쳐의 구성요소로 보고 있다. Bosch[1]의 PLA는 시스템과 시스템의 환경을 인터페이스로 정의한 참조 컨텍스트 외에 아키텍쳐를 분해하는 과정에서 생성된 컴포넌트와 컴포넌트간 관계를 표현하는 구조(Structure)가 아키텍쳐의 구성요소이다. COPA[11]와 QADA[6]는 아키텍쳐의 구성요소를 컴포넌트와 컴포넌트간 관계로만 명시하는 것 외에 뷰도 고려한다. KobrA[12] 방법론에서는 구체적으로 아키텍쳐라고 명시되어 있지 않지만, 컨테인먼트 트리와 컨텍스트 실현으로 유추할 수 있고, 구성요소는 컴포넌트와 컴포넌트간 관계로 이루어져 있다.

그러나, 일부 PLE 방법론의 PLA는 SEI에서 제시한 아키텍쳐나 IEEE, P1471 아키텍쳐에서 중요시하는 뷰

타입과 스타일[13,14]을 구성요소에 포함시키지 않는다. 사용자의 기능적인 요구사항 외 비기능적인 요구사항(품질 요소)을 효과적으로 설계하기 위해 소프트웨어 아키텍처는 뷰와 스타일을 사용한다. 아키텍처를 사용자의 요구에 맞는 특정한 시각으로 묘사하기 위해 뷰타입을 선택하고, 각 뷰타입에 존재하는 여러 스타일 중 하나를 선택하여 소프트웨어 아키텍처가 설계된다. SEI에서는 뷰타입을 모듈 뷰타입, 컴포넌트와 커넥터 뷰타입, 뷰타입, 할당 뷰타입으로 분류한다.

기존 제품 계열 공학에서 제시된 아키텍처의 구성요소와 소프트웨어 아키텍처의 구성요소의 비교 결과로부터 본 논문에서 제시할 PLA의 구성요소를 기존 제품 계열 공학에서 제시된 아키텍처에서 유도된 컴포넌트, 컴포넌트간 관계, 결정 모델과 일반 소프트웨어 아키텍처에서 유도된 뷰, 스타일로 정의한다.

3. 전체 프로세스

본 장에서는 PLA의 구축을 위한 전체 프로세스를 제시한다. 그림 1과 같이 아키텍처 설계 프로세스는 도메인 분석을 통해 추출된 C&V모델과 함께 프로덕트라인 요구사항 명세서(Product-line Requirement Specification, PRS)를 입력으로 받으며, 아키텍처의 설계결과는 컴포넌트 디자인 단계에 영향을 준다. 전체 프로세스는 아키텍처 드라이버(Architectural Driver) 정의, 아키텍처 스타일(Architectural Style) 선택, 아키텍처 스타일 구현, 아키텍처 스타일 통합 및 정제의 4개 활동으로 수행된다. 각 활동은 각각의 입력물과 출력물을 가지며 C&V 모델은 모든 활동들의 입력물로 사용된다.

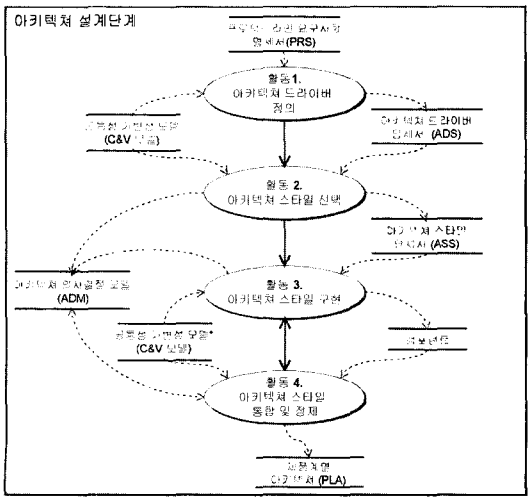

그림 1 프로덕트라인 공학 아키텍처 설계를 위한 프로세스 및 산출물

활동 1의 아키텍처 드라이버의 정의에서는 C&V 모델과 함께 PRS를 입력물로 받아 아키텍처에 중요한 영향을 미치는 아키텍처 드라이버를 추출하는 단계로서 아키텍처 드라이버 명세서(Architectural Driver Specification, ADS)가 정의된다. 이를 이용하여 활동 2에서는 아키텍처 드라이버에 해당하는 아키텍처 스타일을 선택하여 아키텍처 스타일 명세서(Architectural Style Specification, ASS)를 작성한다.

또한 이 단계에서는 아키텍처 드라이버의 가변적인 부분으로 도출된 스타일 가변성이 명시되는 아키텍처 의사결정 모델(Architecture Decision Model, ADM)에 작성된다. 선택된 아키텍처 스타일을 적용하여 활동 3에서는 실질적인 컴포넌트와 컴포넌트간의 연관관계로 아키텍처 스타일을 구현한다. 이를 통해 컴포넌트들이 추출되며 구현된 아키텍처 스타일들은 활동 4의 통합단계를 거쳐 하나의 아키텍처로 표현되고 정제되어 PLA가 도출된다. 활동 3과 4의 경우는 서로 목표가 다르므로 다른 활동으로 구분되었으나 프로세스 진행시는 동시에 수행될 수 있다. 즉, 스타일을 구현하고 이미 구현된 스타일과 통합하며 다시 새로운 스타일 구현시 기존의 통합된 부분적인 아키텍처를 반영할 수 있다.

4. 활동 및 지침

4.1 활동 1. 아키텍쳐 드라이버 정의

아키텍처 드라이버란 아키텍처 설계에 중요하게 영향을 미치는 요구사항을 의미한다[1,15,16]. 따라서 정확한 아키텍처 드라이버를 선택하는 것은 좋은 품질의 아키텍처를 설계하는데 전제 요건이 된다. 활동 1은 이러한 아키텍처 드라이버를 정의하는 단계이다. PLA에는 가변성이 있으므로[9], 아키텍처 드라이버는 공통적인 드라이버와 가변적인 드라이버로 구분될 수 있다.

• 입출력 산출물

활동 1의 입력물은 PRS와 C&V 모델이다. 그림 2에서 보는 바와 같이, PRS는 기능적인 요구사항과 비기능적인 요구사항으로 구분된다. 프로덕트라인 공학 명세서의 분석모델인 C&V 모델은 공통 휘쳐와 가변적인 휘쳐들로 구분되며 휘쳐 트리[4] 또는 테이블 형태[17]로 표현 될 수 있다. 특히 가변적인 휘쳐의 경우 그 형태가 선택적 휘쳐(Optional Feature) 또는 대안휘쳐(Alternative Feature)로 구분된다.

본 논문에서는 PRS와 C&V 모델은 이미 생성되어 이용 가능한 산출물로 정의한다. 또한 PRS 에서 기능적인 요구사항과 비기능적인 요구사항은 구별되므로[18], 기능적인 요구사항과 비기능적인 요구사항의 구분이 가능함을 가정한다.

활동 1의 출력물은 아키텍처 드라이버와 그들의 우선

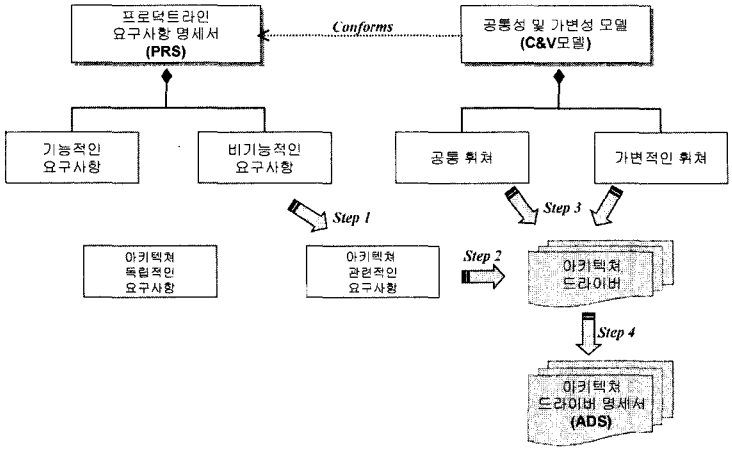

그림 2 프로덕트라인 요구사항에서 PLA로의 정보 흐름

순위를 명세한 ADS이다.

• 세부 지침

활동 1은 그림 2와 같이 네 개의 스텝으로 구성된다. 첫째 스텝은 PRS로부터 아키텍쳐 설계와 관련된 비기능적인 요구사항을 도출하는 것이다. 아키텍쳐 설계에는 비기능적인 요구사항과 기능적인 요구사항이 함께 설계된다. 그러나 아키텍쳐의 설계를 위한 의사결정시는 상세한 기능적인 요구사항보다는 비기능적인 요구사항이 정책적인 의사결정을 하는데 영향을 준다[19].

둘째 스텝은 아키텍쳐 관련 요구사항을 분석하여 아키텍쳐 드라이버를 도출하고 항목화하는 것이다. 각 아키텍쳐 드라이버는 품질 요구사항과 연계되어 정의되며 이어질 다른 활동에서 참조된다.

세번째 스텝은 C&V 모델을 참조하여 도출된 아키텍쳐 드라이버를 공통 아키텍쳐 드라이버와 가변적인 아키텍쳐 드라이버로 구별하는 것이다. 가변적인 아키텍쳐 드라이버의 경우 선택과 대안의 두 가지 형태로 구분할 수 있다.

ADS는 표 1과 같이 아키텍쳐 드라이버 명, 설명, 가변성 형태를 명세한다.

표 1 프로덕트라인 아키텍쳐 드라이버 명세서(ADS)

No	드라이버 명	설명	가변성 형태
1			선택 또는 대안
...	...	...	...

4.2 활동 2. 아키텍쳐 스타일 정의

아키텍쳐 스타일은 아키텍쳐를 구성하는 구성요소와 그들간의 연관관계를 특정 조건에 만족 하도록 설계한 것으로서 아키텍쳐 부분으로 사용할 수 있다[14]. 분석 패턴, 설계 패턴과 같이 아키텍쳐 스타일의 사용은 아키텍쳐 설계에 일반적이며 이는 아키텍쳐 설계에 효율성을 높인다[14,20]. 활동 2는 아키텍쳐 드라이버를 만족시키는 아키텍쳐 스타일을 정의하기 위한 활동이다.

• 입출력 산출물

적합한 아키텍쳐 스타일을 정의하기 위한 입력물로서 ADS가 필요하며 이를 이용하여 선택된 아키텍쳐 스타일과 그 가변성 정보를 ASS와 ADM에 명세한다. ASS에는 아키텍쳐뷰, 아키텍쳐 스타일명과 스타일을 선택한 근거를 명세하며 가변적인 아키텍쳐 드라이버로부터 도출된 아키텍쳐 스타일의 경우 해당 정보를 ADM에 명세한다. ADM은 아키텍쳐에 관련된 모든 가변성 정보를 포함하므로 한 아키텍쳐뷰에 대한 스타일 집합에서의 가변성은 ADM에 명세 되는 가변성 정보의 부분이다.

• 세부 지침

활동2는 아키텍쳐뷰 선택, 아키텍쳐 스타일 선택, 아키텍쳐 스타일 집합 가변성 정의의 세가지 세부 스텝으로 구성된다. 첫째 스텝은 PLA를 설계할 아키텍쳐뷰를 선택하는 것이다. 아키텍쳐를 효과적으로 설계하기 위해 아키텍쳐뷰의 사용은 일반적이다[13,14]. 아키텍쳐뷰란 아키텍쳐를 명세하기 위해 아키텍쳐를 바라보는 시각이다. 아키텍쳐를 설계하기 위해서는 제시된 아키텍쳐뷰 중 적어도 하나의 뷰는 선택되어야 한다[14]. 아키텍쳐뷰는 여러 시각으로 정의되어 있으나 표준으로 사용되는 아키텍쳐 뷰는 아직 정의되지 않았다[21]. 그러나 일반적으로 논리적 기능단위의 모듈뷰, 실행시 모듈과의 연관관계를 나타내는 컴포넌트와 커넥터뷰, 그리고 모듈이 하드웨어 장치에 할당되는 할당뷰의 세가지 시작으로 정리할 수 있다[6,7,14]. 아키텍쳐뷰와 아키텍쳐 스타

일의 관계는 1:N의 관계이다[14]. 즉, 하나의 아키텍처 뷰에 여러 가지 아키텍처 스타일을 표현할 수 있다. 또한 아키텍처 드라이버와 아키텍처뷰와의 관계는 N:M이다. 따라서 하나의 아키텍처 드라이버를 두 개 이상의 아키텍처뷰에 표현할 수 있고 한 아키텍처뷰에 두 개 이상의 아키텍처 드라이버를 설계할 수 있다.

둘째 스텝은 선택된 아키텍처뷰에 표현해야 할 아키텍처 스타일을 선택하는 것이다. 아키텍처 스타일 선택을 위해 활동1에서 정의된 아키텍처 드라이버가 주요하게 참조된다. 예를 들어 개발하고자 하는 시스템이 유지보수를 중요하게 생각하여 모듈화를 강조한다면 아키텍처 드라이버 명은 유지보수성으로 명할 수 있고 유지보수성을 위한 아키텍처 스타일을 모듈뷰에서는 분할스타일(Decomposition Style)을 선택할 수 있다. 그러면, 분할 스타일을 통해 소프트웨어의 기능 모듈들이 어떤 포함관계를 가지는지 모듈간의 독립성을 가지는지가 표현되므로 소프트웨어 유지 보수시 모듈화된 컴포넌트의 연관관계를 고려하여 사용할 수 있다. ASS는 다음의 표 2와 같이 표현할 수 있다.

표 2 아키텍처 스타일 명세서(ASS)

아키텍처뷰	아키텍처 드라이버 명	아키텍처 스타일	근거

세번째 스텝은 아키텍처 드라이버의 가변적인 항목에 대해 아키텍처 스타일에 반영하고, ADM을 명세 하는 것이다. 소프트웨어 아키텍처는 여러 아키텍처뷰로 보여질 수 있고 한 아키텍처뷰를 통해 그 뷰에 대한 소프트웨어 전체를 볼 수 있다. 아키텍처뷰에 의해 표현된 아키텍처에는 여러 아키텍처 드라이버로부터 유도된 여러 아키텍처 스타일들이 있다. 그런데 아키텍처 드라이버에 가변성이 존재하므로 아키텍처 스타일 또한 아키텍처 전체의 시각에서 보면 가변적일 수 있다. 이렇게 한 뷰에서 표현된 아키텍처 스타일에 대한 가변성을 아키텍처 스타일 집합 가변성(Architecture Style Set Variability)라 정의하며 그림 3과 같이 표현할 수 있다.

그림은 아키텍처 드라이버 집합과 아키텍처 스타일 집합으로 구분된다. 아키텍처 드라이버 집합에서 가변적 아키텍처 드라이버 i는 아키텍처 드라이버 i-a와 i-b로 선택될 수 있다. 따라서 이로부터 도출된 아키텍처 스타일 집합은 {아키텍처 스타일 a, 아키텍처 스타일 b, ···, 아키텍처 스타일 i-a 또는 아키텍처 스타일 i-b, ..}로 정의되며 아키텍처 스타일 i가 가변적이다.

이러한 아키텍처 스타일 집합에서의 가변성은 표 3에서와 같이 ADM에 명세 된다. 표 3의 ADM에는 아키텍처 스타일 집합 가변성만 명세 되었으나 이어지는 활동을 통해 추가적인 아키텍처 가변성이 명세 된다.

아키텍처 의사결정 모델은 아키텍처에서 가변적인 내용이 명세 되어 있으며 가변점, 가변치, 가변성타입, 그리고 효과(Effect)와 수행작업(Task)으로 구성된다[23]. 효과란 어플리케이션 생성시 가변점에서 하나의 가변치가 선택되고 인스턴시에이션 활동이 마친 후의 최종 핵심자산의 상태를 명세한다. 따라서 이 부분에서 가변성간의 연관관계를 명세할 수 있으며 이에 따라 인스턴시에이션 작업시 수행되어야 할 활동 리스트를 수행작업 항목에 명세 할 수 있다.

4.3 활동 3. 아키텍처 스타일 구현

활동 2로부터 아키텍처뷰들과 이에 대한 아키텍처 스타일 리스트를 도출하였다. 활동 3에서 항목화 된 아키텍처 스타일들을 구체적인 컴포넌트와 컴포넌트간의 연

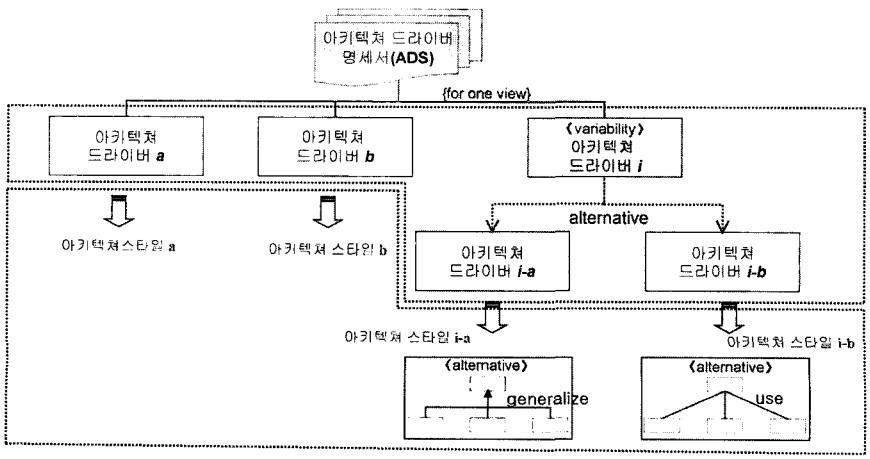

그림 3 PLA에서의 아키텍처 스타일 집합 가변성

표 3 아키텍처 의사결정 모델(ADM)에서의 아키텍처 스타일 집합 가변점

가변점	가변치		가변성 타입	효과(Effect)	수행작업(Task)
	아키텍처 드라이버	아키텍처 스타일			
아키텍처 스타일 집합			선택 또는 대안		
	...	...	...	...	...
...	...	...	...	...	...

관관계를 적용하여 표현한다.

• 입출력 산출물

아키텍처 스타일 집합과 그에 대한 가변성을 명세한 ASS와 ADM을 입력물로 받아 컴포넌트와 컴포넌트간의 연관관계로 표현된 아키텍처 스타일들이 도출된다. 그 과정을 통해 ADM에는 아키텍처 스타일 내에 컴포넌트 또는 그들간의 연관관계에 대한 가변성이 추가 명세 된다.

• 세부지침

아키텍처 스타일 구현은 컴포넌트 추출, 적용, 가변성 해결의 세가지 스텝으로 수행된다. 아키텍처는 컴포넌트와 그들간의 연관관계로 정의 된다. 여기에서 컴포넌트란 아키텍처를 구성하는 구성 단위로서 소프트웨어 기능을 위한 논리적인 모듈 또는 데이타베이스관리 시스템(DBMS), 서버 등의 하드웨어의 장치가 될 수 있다 [22].

아키텍처 스타일을 구현하기 위한 첫째 스텝은 아키텍처의 컴포넌트를 추출하는 것이다. 논리적인 모듈을 추출하기 위해 [23]에서 제시된 컴포넌트 추출 기법을 참조 할 수 있다. 하드웨어 장치는 C&V모델이나 PRS 등의 프로덕트라인의 정책이나 비기능적인 요구사항으로부터 추출 조건을 도출 할 수 있다.

둘째 스텝은 추출된 컴포넌트를 스타일에 적용하는 것이다. 각각의 아키텍처 스타일에 대해 관련 컴포넌트를 할당하고 연관관계를 도출한다. 스타일을 구현하는 컴포넌트와 그의 연관관계에 따라 의존(Dependency), 포함(Composition), 연관(Association)등의 관계들이 확장되어 표현될 수 있다.

셋째 스텝은 ADM에 스타일 내의 가변성 정보를 추가하는 것이다. C&V 모델에 따라 기능적/비기능적 휘처로부터 유도된 모듈들 중에는 모듈 자체가 선택 또는 대안의 가변성을 그림 4와 같이 가질 수 있다[22].

이러한 가변성 정보들은 표 4과 같이 ADM에 가변점

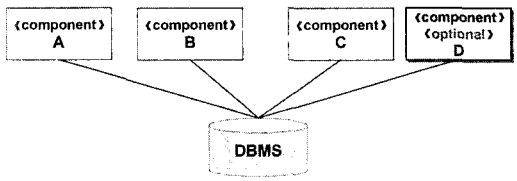

그림 4 PLA에서의 아키텍처 스타일 가변성

이 아키텍처 스타일인 가변성으로 표현될 수 있다.

4.4 활동 4. 아키텍처 스타일 통합 및 정제

이 단계는 PLA를 설계하는 마지막 단계로서 구현된 여러 아키텍처 스타일들을 통합하여 한 아키텍처뷰에 대한 하나의 아키텍처를 생성하고 통합시 도출되는 문제들을 해결하여 아키텍처를 정제하는 단계이다. 통합시 도출되는 문제는 스타일이 겹쳐지는 부분에 대한 가변성의 해결이 대표적이다. 따라서 이러한 정보가 ADM에 추가되며 이를 통해 ADM의 효과와 수행작업을 정제한다.

• 입출력 산출물

구현된 아키텍처 스타일들과 ADM을 입력물로 받아 활동 4를 통하여 통합된 PLA를 도출된다. 또한 최종 아키텍처 가변성 모델로서 ADM을 정제한다.

• 세부지침

아키텍처 스타일 통합 및 정제는 두 스텝으로 수행된다. 첫째 스텝은 한 아키텍처뷰에 대한 구현된 스타일들을 정규화 하는 것이다. 아키텍처 스타일에 따라 구현된 정도는 그 상세도가 다를 수 있다. 따라서 본 스텝에서는 모아진 스타일들의 통합을 고려하여 각 스타일에 포함된 컴포넌트들의 크기(Granularity)를 정규화 해야 한다. 이를 통해 스타일의 컴포넌트들은 분할 되어 상세화 될 수 있다.

둘째 스텝은 스타일들을 연결하여 중복되는 부분을 추출하여 통합하는 단계이다. 그림 5는 두 개의 아키텍처 스타일의 통합시 나타나는 중복영역을 나타낸다.

그림 5에서 C, D, E 모듈은 한 데이타를 공유하는 공

표 4 아키텍처 의사결정 모델 (ADM)에서의 아키텍처 스타일 가변점

가변점	가변치	가변성 타입	효과(Effect)	수행작업(Task)
...	...	...	...	...
아키텍처 스타일	컴포넌트 또는 컴포넌트간의 연관관계	선택 또는 대안		

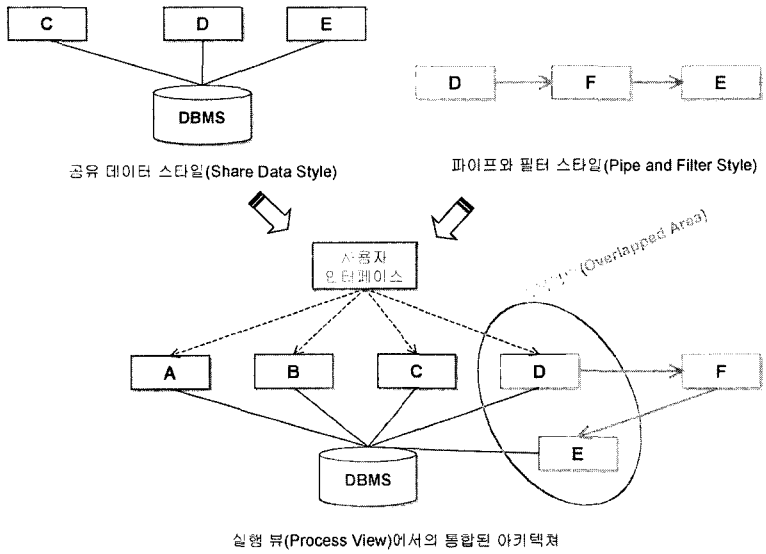

그림 5 통합된 아키텍처에서의 스타일 중복영역(Overlapped Area)

유 데이타 스타일을 적용되어 표현되었고, D, E, F 모듈에는 파이프와 필터 스타일이 적용되었다. D와 E 모듈은 두 스타일에 모두 중복으로 표현된 모듈이다. 그래서, 두 스타일을 하나로 통합할 경우 같은 중복 영역에 있는 D와 E 모듈은 하나의 모듈로 표현되어야 한다.

이러한 중복영역은 가변성을 포함한 영역과 가변성을 포함하지 않은 영역으로 구분된다. 가변성을 포함하지 않은 영역은 중복된 모듈 중 하나의 모듈만 표현한다. 그러나 가변성을 포함한 영역은 가변성 타입에 따라 아키텍처에 적용되는 형태를 상세하게 표현하여야 한다. 그림 6은 가변성을 포함한 아키텍처 스타일 중복 영역을 나타낸다. D 모듈은 D'와 D"로 분류되고, D"은 선택 가능한 가변치로 D 모듈에 가변성이 존재한다고 하자. 따라서 그림 5의 D 모듈은 D'와 D"로 나뉘어 지며 파이프와 필터 스타일과의 연결은 D"로 연결된다. 또한 F와 E 모듈은 D"가 선택되었을 경우만 연관관계를 가지므로 선택적인 관계(«optional»)로 정제한다.

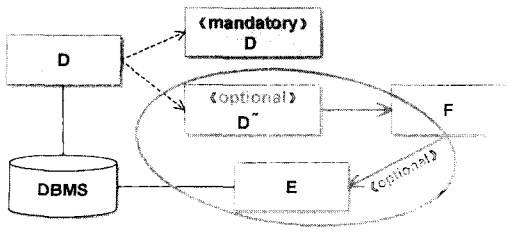

그림 6 가변성을 포함한 아키텍처 스타일 중복(Overlap) 영역

위와 같은 정제단계를 거쳐 변경되는 정보는 ADM에 추가되어야 하며 특히 영향과 수행작업 부분에서 상세하게 표현되어야 한다.

5. 사례연구

제안된 프로세스의 적용을 위해 도서판매 도메인의 사례연구를 다음과 같이 수행하였다. 도서판매 도메인에서는 서점의 구분을 온라인 서점과 오프라인 서점으로 구분 한다. 각 서점은 크게 재고관리, 판매관리, 회계관리, 고객관리로 구분되며 서점에 따라 온라인 서점을 병행할 수 있다. 온라인 서점인 경우 온라인 주문 및 결제 관리의 추가 관리가 필요하다.

5.1 활동 1. 아키텍쳐 드라이버 정의

도서판매 도메인으로부터 표 5와 같은 품질 관련 요구사항을 도출한다. 데이타의 무결성과 확장성은프로덕트라인에서 공통적인 요구사항이므로 가변성을 가지지 않아 '필수'로 표현하였으며 보안성은 온라인 서비스 기능을 선택하였을 경우 추가되는 품질 요구사항으로 가변성 형태는 '선택'이다.

5.2 활동 2. 아키텍쳐 스타일 정의

활동 1에서 정의된 아키텍쳐 드라이버에 따른 ASS를 표 6과 같이 명세한다. 아키텍쳐뷰는 지면의 제한으로 실행시 모듈간의 연관관계를 나타내는 컴포넌트와 커넥터 뷰만을 나타낸다.

도출된 아키텍쳐 스타일 집합은 {공유데이타 스타일, 계층 스타일} 또는 {공유데이타 스타일, 계층 스타일, 파이프와 필터 스타일}로 구분되며 파이프와 필터 스타

표 5 도서판매 프로덕트라인의 아키텍쳐 드라이버 명세서

No	드라이버 명	설명	가변성 형태
1	데이타의 무결성	모든 기능 모듈은 일관적인 데이타를 공유해야 한다.	필수
2	확장성	사용자 인터페이스의 변경에도 비즈니스 로직은 영향을 받지 말아야 한다.	필수
3	보안성	온라인 결제시 결제자의 신용에 대한 인증이 필요하다.	선택

표 6 도서판매 프로덕트라인의 아키텍쳐 스타일 명세서(ASS)

아키텍처뷰	아키텍처드라이버명	아키텍쳐 스타일	근거
컴포넌트와 커넥터뷰 (실행뷰)	데이타의 무결성	공유데이타 스타일	각 기능모듈이 데이타를 공유 한다.
	확장성	계층 스타일	사용자 인터페이스 모듈과 비즈니스 로직 모듈이 독립적이다.
	보안성	파이프와 필터 스타일	결제 인증된 결제결과만 회계처리 한다.

표 7 도서 판매 PLA 의사결정 모델 (ADM) 에서의 아키텍쳐 스타일 집합 가변점

가변점	가변치		가변성 타입	가변치	효과(Effect)	수행작업(Task)
	아키텍쳐 드라이버	아키텍쳐 스타일				
아키텍쳐 스타일 집합	보안성	파이프와 필터스타일	선택	선택	온라인 결제가 외부 인증시스템과 연결된다.	1. 외부 인증시스템을 선택한다. 2. 온라인 결제시스템과 인터페이스를 맞춘다.
				제외	온라인 결제 기능이 제거된다.	1. 온라인결제 기능과 관련된 설계 제거한다.

일은 선택 가능한 스타일이다. 이러한 가변성 정보를 표 7과 같이 표현한다.

5.3 활동 3. 아키텍쳐 스타일 구현

활동 2에서 도출된 아키텍쳐 스타일을 구현하기 위해 기능의 의존성과 응집도에 따라 도서 판매 프로덕트라인의 기능 컴포넌트를 고객, 품목, 판매, 온라인 주문, 배송, 결제, 회계관리로 정의한다. 표 6에서 필수 스타일은 공유데이타 스타일, 계층 스타일이고 파이프와 필터 스타일은 선택적인 스타일로 정의하였으며 그림 7은 그에 대한 아키텍쳐 스타일 구현을 보여준다.

가변성에 대해서는 표 8에서와 같이 온라인 기능에 대한 선택 여부에 따라 온라인 주문 및 배송 컴포넌트가 선택적인 컴포넌트가 되고, 결제 컴포넌트는 현금 결제처리와 온라인 결제 처리 컴포넌트를 대안으로 한다.

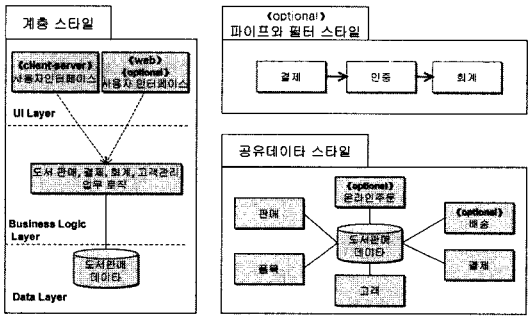

그림 7 도서 판매 PLA에서의 아키텍쳐 스타일 가변성

5.4 활동 4. 아키텍쳐 스타일 통합 및 정제

표 8 아키텍쳐 의사결정 모델 (ADM)에서의 아키텍쳐 스타일 가변점

가변점		가변치		가변성 타입	가변치	효과(Effect)	수행작업(Task)
아키텍쳐 스타일 집합		아키텍쳐 드라이버	아키텍쳐 스타일	-	-	-	-
		보안성	파이프와 필터스타일	선택	선택	온라인 결제가 외부 인증시스템과 연결된다.	1. 외부 인증시스템을 선택 2. 온라인 결제시스템과 인터페이스를 맞춤
					제외	온라인 결제 기능이 제거된다.	1. 온라인결제 기능과 관련된 설계 제거
아키텍쳐 스타일	공유 데이타 스타일	온라인 주문/배송 컴포넌트		선택	선택	온라인 결제컴포넌트기능이 추가된다.	1. 온라인 결제컴포넌트기능이 추가 2. 보안관련 가변성을 확인
					제외	온라인 주문/배공 컴포넌트가 제거된다.	온라인 주문/배송 컴포넌트가 제거
	계층 스타일	웹유저인터페이스 컴포넌트		선택	선택	웹유저인터페이스 컴포넌트를 사용한다.	-
					제외	웹유저인터페이스 컴포넌트 제거된다.	웹 유저인터페이스 컴포넌트를 제거

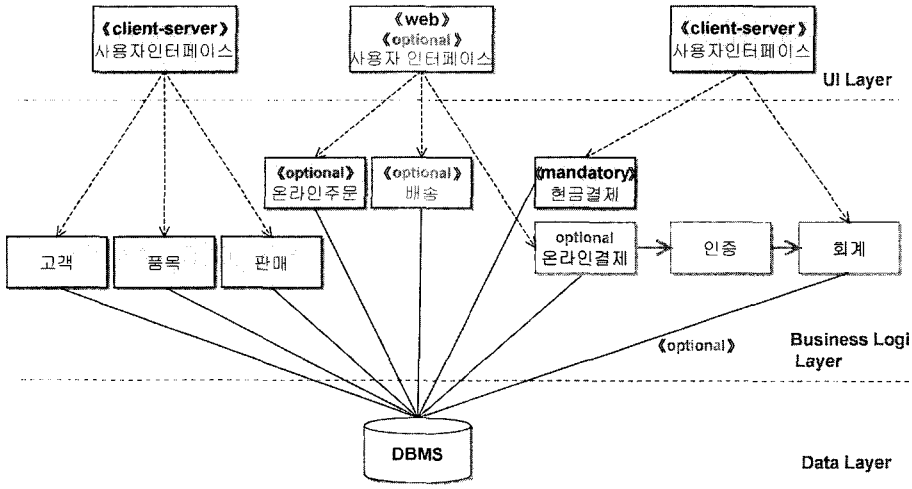

그림 8 도서판매 프로덕트라인의 아키텍처

활동 3을 통해 구현된 아키텍처를 통합하면 그림 8과 같다. 세 개의 아키텍처 스타일이 통합되면서 결제 컴포넌트는 현금 결제와 온라인 결제로 분할 되었으며 파이프라인과 필터 스타일에는 온라인 결제만 적용되는 것으로 아키텍처가 정제 되었다.

6. 결 론

프로덕트라인 공학은 여러 어플리케이션들이 공유할 수 있는 핵심자산을 사용하여 소프트웨어의 생산성을 높이고 재사용성을 증대 함으로써 개발 효율성을 높이는 대표적인 소프트웨어 개발방법으로 수용되고 있다. 프로덕트라인 공학의 개발방법에서 중요한 재사용 단위로서 PLA는 중요한 구성요소 중 하나이다. 본 논문은 PLA를 설계하기 위한 프로세스를 4가지 활동으로 정의하였으며 각 활동의 입력물과 출력물을 정의하였고 활동을 위한 세부 지침을 제안 하였다. 또한 프로덕트라인 공학에서의 가변성의 정의와 표현방법을 아키텍처 수준에서 구체적인 산출물 양식과 함께 표현하였다. 그리고 제안된 프로세스의 실효성을 위해 도서판매 도메인에서의 사례를 연구하였다. 제안된 프로세스와 PLA가변성 정의로 프로덕트라인 공학에 대한 효율적인 실무적 접근을 예상한다.

참 고 문 헌

[1] Bosch, J. *Design and Use of Software Architectures*, Addison-Wesley, 2000.

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

[3] Bayer, J., flege, O., and Gacek, C., "Creating Product Line Architectures," *proceeding of IW-SAPF-3*, LNCS 1951, Springer-Verlag Berlin Heidelberg, 2000.

[4] Kang, K. et. al., "FORM: A Feature-Oriented Reuse Method with Domain-Specific Reference Architectures," Annals of Software Engineering, 5, 1998, pp. 143-168.

[5] Kang, K., Lee, J., and Donohoe, P., "Feature-Oriented Product Line Engineering," *IEEE Software*, Vol. 9, No. 4, Jul./Aug. 2002, pp. 58-65.

[6] Matinlassi, M., Niemela, E., and Dobrica, L., "Quality-driven architecture design and quality analysis method : A revolutionary initiation approach to a product line architecture," VTT Technical Research Center of Finland, ESPOO 2002, 2002.

[7] Ceron, R., Arciniegas, J., Ruiz, J., Cuenas, J., Bermejo, J., and Capilla, R., "Architectureal Modeling in Product Family Context," *proceeding of EWAS, LNCS 3047*, Springer-Verlag Berlin Heidelberg, 2004.

[8] Thiel, S., "On the Definition of a Framework for an Architecting Process," *proceeding of PFE-4, LNCS 2290*, Springer-Verlag Berlin Heidelberg, 2002.

[9] Thiel, S., and Hein, A., "Systematic Integration of Variability into Product Line Architecture Design," *proceeding of SPLC2, LNCS 2379*, Springer-Verlag Berlin Heidelberg, 2002.

[10] Bayer, J. et al., "PuLSE: A Methodology to Develop Software Product Lines," *Proceeding of Symposium on Software Reusability '99*, May 1999.

[11] Obbink, H. et al., "COPA: A Component-Oriented Platform Architecting Method for Families of Software-Intensive Electric Products," *Tutorial for*

the First Software Product Line Conference (SPLC1), Aug. 2000.
[12] Atkinson, C., et al., *Component-based Product Line Engineering with UML*, Addison Wesley, 2001.
[13] IEEE Recommended Practice for Architectural Description of Software-Intensive Systems (IEEE Standard P1471); IEEE Architecture Working Group (AWG); 2000.
[14] Clements, P., et al., *Documenting Software Architectures Views and Beyond*, Addison-Wesley, 2003.
[15] Perry, D., "Generic Architecture Descriptions for Product Lines," *proceeding of ARES, LNCS 1429*, Springer-Verlag Berlin Heidelberg, 1998.
[16] America, P., et al., "Scenario-Based Decision Making for Architectural Variability in Product Families," *proceeding of SPLC 2004, LNCS 3154*, Springer-Verlag Berlin Heidelberg, 2004.
[17] Choi, S., Chang, S, and Kim, S., "A Systematic Methodology for Developing Component Frameworks," *proceedings of the 7th Fundamental Approaches to Software Engineering Conference, LNCS 2984*, Springer-Verlag Berlin Heidelberg, 2004.
[18] Lauesen, S., *Software Requirements Styles and Techniques*, Addison-Wesley, 2002.
[19] McGovern J., et al., *A Practical Guide to Enterprise Architecture*, Prentice Hall, 2003.
[20] Garlan, D., Allen, R., Ockerbloom, J., "Exploiting Style in Architectural Design Environments," *Proceedings of SIGSOFT'94*, Foundations of Software Engineering, pp. 175-188, 1994.
[21] Woods, E., "Experiences Using Viewpoints for Information Systems Architecture: An Industrial Experience Report," *proceeding of EWSA 2004, LNCS 3047*, Springer-Verlag Berlin Heidelberg, 2004.
[22] Gomma, H., *Designing Software Product Lines with UML from Use Cases to Pattern-based Software Architectures*, Addison-Wesley, 2004.
[23] Kim, S. and Chang, S., "A Systematic Method to Identify Component," *Proceedings of APSEC 2004*, Nov., 2004.

장 수 호
2003년 숭실대학교 컴퓨터학부 공학사. 2005년 숭실대학교 컴퓨터학과 공학석사. 2005년~현재 숭실대학교 컴퓨터학과 박사과정. 관심분야는 제품계열 공학(PLE), 소프트웨어 아키텍처, 모델 기반 아키텍처(MDA)

라 현 정
2003년 경희대학교 우주과학과 이학사. 2004년~현재 숭실대학교 컴퓨터학과 석사과정. 관심분야는 제품계열 공학(PLE), 소프트웨어 아키텍처, 모델 기반 아키텍처(MDA)

김 수 동
1984년 Northeast Missouri State University 전산학 학사. 1988년/1991년 The University of Iowa 전산학 석사/박사. 1991년~1993년 한국통신 연구개발단 선임연구원. 1994년~1995년 현대전자 소프트웨어연구소 책임연구원. 1995년 9월~현재 숭실대학교 컴퓨터학부 부교수. 관심분야는 객체지향 S/W공학, 컴포넌트 기반 개발 (CBD), 제품계열 공학(PLE), 모델 기반 아키텍처(MDA)