프로덕트 라인 소프트웨어 개발 프로세스†

포항공과대학교 이재준 · 강교철*

1. 서 론

소프트웨어 재사용의 초점은 소스 코드 재사용에서 소프트웨어 설계의 재사용으로, 설계 재사용에서 다시 영역 공학(domain engineering)에 기반을 둔 재사용으로 그 중심이 옮겨져 왔다. 이러한 변화의 이유는 소스 코드를 재사용하기 위해서는 설계의 재사용이 선행되어야 했고, 또한 설계를 재사용하기 위해서는 영역 공학을 통해서 어떤 영역에서 공통적으로 쓰이는 영역 자산(domain asset)을 먼저 개발하여야 했기 때문이다. 따라서 지난 수년 간 소프트웨어 재사용 연구는 영역 분석 및 공학(domain analysis and engineering)에 중점을 두고 수행되어 왔다[1, 2, 3, 4, 5, 6, 7].

영역 공학적인 접근 방법은 해당 응용 영역에 대해서 광범위한 분석을 실시하고 이를 통하여 재사용성이 높은 컴포넌트를 개발하는 것이다. 그러나 이러한 접근방법은 많은 경우에 영역 내의 모든 프로덕트를 대상으로 포함하려는 경향이 있었으며, 이로 인하여 개발 비용과 기간에 관련된 문제가 발생하기도 하였다[8](그림 1 좌측 부분 참조). 또한 재사용성을 극대화하기 위한 노력으로 소프트웨어 구조의 계층화(layering)와 정보 은닉을 위한 캡슐화(encapsulation for information hiding)를 강조하다보면, 결과적으로 성능이 떨어지는 시스템을 개발하게 되는 경우도 있었다[9]. 이와 같이, 어떤 영역을 중심으로 재사용을 하려함에 있어서 적절한 영역의 범위와 재사용성의 정도를 결정하는 것은 항상 어려운 문제이었다[10].

"프로덕트 라인 소프트웨어 공학"(product line software engineering)은 위에서 언급한 문제 가운데 일부를 해결하고자 최근에 제시된 패러다임이다[11, 12, 13, 14]. 이 패러다임 하에서 소프트웨어 재사용은 우연한 것이 아니라, 특정 프로덕트 라인을 재사용하기 위해서 신중하게 수립된 계획 하에 이루어진다. 프로덕트 라인에서도 영역 분석을 실시하나, 프로덕트 라인을 위한 영역 분석은 명확한 목표가 있다. 즉, 재사용 가능한 컴포넌트를 기반으로 세워진 프로덕트 라인 환경 하에서 프로덕트를 개발할 수 있도록 하는 것과, 프로덕트 라인의 확장 및 변화 가능성 그리고 기술의 발전을 수용할 수 있도록 하는 것이다(그림 1의 우측 부분 참조). 이렇게 명확한 목표는 영역 분석을 해야 하는 범위뿐만 아니라, 이를 기반으로 하는 프로덕트 라인 자산(product line asset)의 개발 범위도 명확하게 해준다. 따라서 프로덕트 라인 자산 개발과 프로덕트 개발 프로세스는 기존의 영역 공학적인 접근방법보다 효율적이고 효과적으로 수행된다.

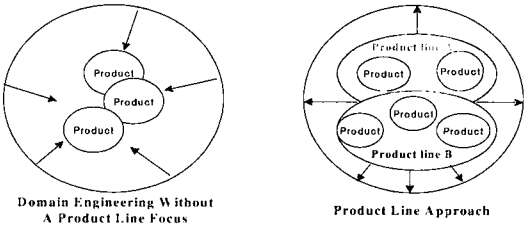


그림 1 재사용을 위한 영역 분석의 범위

어떤 프로덕트 라인의 핵심이 되는 소프트웨어 자산(software asset)을 개발하기 위해서 프로덕트 라인 방법은 해당 프로덕트 라인의 공통점을 찾아내고 차이점을 관리할 수 있는 방안을 제시하여야 한다. 휘처(feature) 중심의 영역 분석 방법인 Feature Oriented Domain Analysis method(FODA)[2]는

† 이 연구는 교육부에서 지원한 BK21 프로그램의 연구비 지원에 의해서 연구되었습니다.

* 종신회원

1990년 Software Engineering Institute를 통하여 소개된 이후, 산업계(예 : Italia Telecom [15], Northern Telecom[16], Hewlett Packard Company[17, 18], Robert Bosch GmbH[19] 등)와 학계 (예 : Software Engineering Institute[20, 21], Technical University of Ilmenau[22], 포항공대[23, 24, 25, 26] 등)에서 영역 분석과 프로덕트 라인을 분석하는 방법으로 광범위하게 사용되어 왔다. 뿐만 아니라, 많은 영역 분석 방법들이 훼처 중심의 분석 방법을 이용하여 영역의 공통점과 차이점을 분석하고 있다(예 : ODM[6], FeatureRSEB[17], FODAcom[15], DEM RAL[22] 등).

포항공대 소프트웨어 공학 연구실에서도 FODA를 기반으로 프로덕트 라인 방법론을 개발하였으며, 이 방법론이 Feature-Oriented Reuse Method (FORM)이다. FORM은 소프트웨어 아키텍처 설계[23]와 객체 지향 컴포넌트 개발[25]을 지원하며, 이 방법은 다양한 산업계 프로덕트 라인(예 : 전자 게시판[23], 사설 교환기[24], 엘리베이터 컨트롤[25, 26] 등)에 적용되어, 프로덕트 라인 소프트웨어 개발 환경과 소프트웨어 자산을 개발하는데 적용되었다. 특히, 마케팅 & 프로덕트 계획(marketing and product plan : MPP)을 프로덕트 라인 분석 및 자산 개발의 핵심요소로 사용하도록 FORM 방법론을 확장[27]함으로서, 비즈니스 영역과 엔지니어링 영역을 연결하는 방안을 제시하게 되었다. 본 논문에서는 이 확장된 FORM 방법론을 중심으로 프로덕트 라인 개발 프로세스를 설명한다.

본 논문의 구성은 다음과 같다. 다음 장에서 프로덕트 라인 개발 프로세스의 전체적인 구성을 살펴보고 이를 구성하는 세부 활동에 대해서 3장과 4장에서 각각 설명한다. 5장에서는 결론으로 프로덕트 라인 방법을 적용할 때 고려해야 할 사항을 논의한다.

2. 프로덕트 라인 소프트웨어 개발 프로세스

FORM 방법론의 핵심은 프로덕트 라인을 훼처 중심으로 분석하고 이러한 훼처를 이용하여 재사용 가능하고 변화에 적응시킬 수 있는 프로덕트 라인 자산을 개발하는데 있다. 여기서 우선 훼처의 의미를 살펴보도록 하자. 어떤 응용 프로그램 영역(예 : 전자 게시판 시스템 영역, 교환기 시스템 영역 등)이 성숙됨에 따라 이들 응용 프로그램을 구현하기 위한 표준 기술이나 사용자와 개발자 사이에 의사소통을 위해 사용되는 표준 용어 등이 나타나는 것을 알 수 있다. 예를 들면, 교환기 시스템 영역에서 사용되는 International Telecommunication Union Telecommunication Standardization Sector(ITU-T)의 표준화된 권고안이나, 전자 게시판 시스템 영역의 "ANSI," "VT-100"과 같은 터미널 운용 기술 등이 그것들이다. 이러한 표준화된 용어들은 해당 프로덕트 라인의 전문가들이 그들의 아이디어, 요구사항, 그리고 구현 방법 등에 대하여 의사소통을 하기 위한 영역 용어(domain terminology)로서 사용된다. 이때 사용되는 영역 용어들을 훼처라고 하며, 이를 모델화 한 것이 훼처 모델(feature model)이다[2].

FORM 방법론은 크게 두 가지의 프로세스로 이루어져 있다 : 프로덕트 라인 자산 개발 프로세스(product line asset development process)와 프로덕트 개발 프로세스(product development process)이다(그림 2 참조). 프로덕트 라인 자산 개발 프로세스는 마켓을 분석하고 이에 따른 프로덕트 계획을 수립하고 정련하는 마케팅 & 프로덕트 계획 수립 및 정련(marketing and product planning and refinement)활동, 프로덕트 라인을 훼처 위주로 분석하는 훼처 모델링 (feature modeling) 활동, 프로덕트 라인의 요구사항을 분석(product line requirement analysis)하는 활동, 프로덕트 라인에서 사용하게 될 참조 아키텍처를 개발하는 개념적 아키텍처 설계 및 아키텍처 정련(conceptual architecture design and architecture refinement)활동, 그리고 재사용 가능하고 변화에 적응성을 갖는 컴포넌트를 개발하기 위한 설계 객체 모델링 및 컴포넌트 설계(design object modeling and component design)활동으로 구성되어 있다.

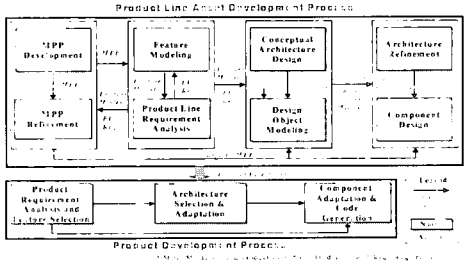


그림 2 FORM 프로덕트 라인 개발 프로세스

프로덕트 개발 프로세스는 프로덕트 라인 자산 개발 프로세스에서 개발된 자산을 이용하여 고객이 원하는 프로덕트를 개발하기 위한 활동으로 구성되어 있다. 고객의 요구사항에 따른 프로덕트의 요구사항 분석과 휘처 모델에서 휘처 선택(product requirement analysis and feature selection)이 이루어지게 되고, 이에 적합한 참조 아키텍처가 선택되고 적응되게(architecture selection and adaptation) 된다. 이러한 기반 위에서 시스템을 구성하게 되는 컴포넌트들을 아키텍처 구조에 맞게 조합하고 적응시키는 과정을 거쳐 최종 프로덕트 코드를 생성(component adaptation and code generation)하게 된다. 이하의 장에서 각각의 프로세스를 구성하고 있는 활동들에 대하여 살펴보도록 한다.

3. 프로덕트 라인 자산 개발 프로세스

3.1 마케팅 & 프로덕트 계획 수립 활동

프로덕트 라인을 위한 소프트웨어 자산을 개발하기 위해서 고려해야 하는 중요한 두 가지 요소는 마켓이 요구하는 휘처를 원하는 시기에 배포할 수 있어야 한다는 것과 마켓의 변화를 프로덕트 라인 자산이 수용할 수 있어야 한다는 것이다. FORM 방법론은 이를 위하여 마켓 정보를 분석하고 체계화한 MPP를 작성한다. 이 MPP는 하나의 계획으로서 그 내용은 마케팅 계획 부분과 프로덕트 계획 부분으로 이루어져 있다[27](그림 3 참조). 마케팅 계획 부분은 마켓 분석(marketing analysis)과 마케팅 전략(marketing strategy)으로 구성되어 있으며, 이를 바탕으로 프로덕트 휘처와 이 프로덕트 휘처를 고객에게 전달하는 방법을 포함하는 프로덕트 계획을 작성하게 된다.

이러한 MPP가 프로덕트 라인 자산을 개발하는데 중요한 역할을 하는 이유는 다음과 같다 :

- 요구사항 분석은 일반적으로 프로덕트 라인이 가져야하는 기능적인 측면에 중점을 두고 있으며 설계 제약사항이나 품질 요구사항(예 : 성능, 유지보수성, 재사용성 등)은 철저히 조사되지 않거나 혹은 개발자에게 명확하게 전달되지 않는 경우가 많다. 이러한 요소들이 MPP를 작성하는 동안 분석되고 문서화되면서, 프로덕트 라인 자산을 개발하는데 중요한 정보를 제공하게 된다. 예를 들면, 새로이 진입하고자 하는 마켓의 예상되는 사용자들의 문화적 혹은 지식적 배경(예를 들면, 생활 습관이나 컴퓨터에 대한 지식 정도 등)은 사용자 인터페이스나 프로덕트 설치 프로그램 등을 개발할 때 반드시 고려해야 할 사항으로서 MPP에 기술되어 개발자에게 전달될 수 있다.
- 다양한 프로덕트를 개발해야 하는 프로덕트 라인의 특성은 하나의 응용 프로그램을 개발하는 것과는 다른 엔지니어링 측면을 가지고 있다. 즉, 하나의 프로덕트 라인은 하나 이상의 마켓 세그먼트(market segment)를 목표로 하므로, 이를 위하여 고려해야 할 사항들이 존재한다. 예를 들어 보안 시스템(security system)의 경우, 가정용 시스템 마켓과 동시에 사무용 시스템 마켓을 목표로 하여 프로덕트 라인을 구성하였다면, 이 두 마켓에 속한 사용자의 서로 상이한 요구사항(예: 가정용에서는 단순히 물체의 움직임을 감지하여 침입을 경보하는 기능이 필요하지만, 사무용에서는 물체의 움직임뿐만이 아니라 온도, 소리를 감지하여 경보 및 퇴로를 차단하는 기능이 필요)과 운용자의 지식 배경(예 : 가정용은 컴퓨터 관련 지식이 없는 일반인이 시스템을 사용하는 반면에, 사무용은 컴퓨터를 전공한 전담 유지보수 요원이 있음) 등을 분석하여 MPP에 반영하여야 하며, 프로덕트 라인 자산은 이를 수용할 수 있도록 개발되어야 한다.
- 하나의 프로덕트에 어떤 휘처가 포함될 것인지 그리고 어느 단계(예 : 배포시, 설치시, 혹은 운용중)에서 프로덕트에 휘처가 포함될 것인지에 대한 결정과 프로덕트를 고객에게 어떤 형태(예: 미리 패키지화하거나 혹은 고객이 직접 선택할 수 있는 형태)로 전달할 것인지에 대한 결정이 MPP를 작성하면서 내려지게 된다. 이러한 결정은 프로덕트 자산을 설계하고 개발하는데 제약사항이

- Marketing Plan
  - Market analysis
    - Market segments
      - *Needs assessment*
      - *Customer profile*
        - *Cultural and technical backgrounds (e.g., computing skill) of users*
        - *Computing environments*
      - *Legal constraints*
    - Assessment of business opportunities
      - *Time to market, price range, etc.*
  - Marketing strategy
    - *Product delivery methods*
    - Other business considerations on sales, pricing, etc.
- Product Plan
  - *Product functional features and quality attributes*

그림 3 마케팅 & 프로덕트 계획의 구성요소[27]

나 고려사항으로 사용된다.

- 프로덕트 자산 개발 활동과 마케팅 활동이 MPP를 통하여 서로 밀접한 관계를 갖게 된다. 따라서 개발자는 마켓의 요구사항을 수용할 수 있는 프로덕트 라인 자산을 설계, 개발하게 되고, 마케팅 부서는 기술적 타당성과 개발 능력 등을 고려하여 마케팅 전략을 수립할 수 있게 된다.

이 MPP는 프로덕트 라인 자산 개발의 기반을 제공하며, 휘처 모델과 프로덕트 라인 요구사항을 이용하여 다시 정련된다. 이제 다음 절에서 휘처 모델링 및 프로덕트 라인 요구사항 분석 활동에 대하여 살펴보자.

3.2 휘처 모델링 및 프로덕트 라인 요구사항 분석

휘처 모델링은 프로덕트 라인의 공통점과 차이점을 분석하고 이를 휘처 모델을 이용하여 나타내는 활동을 말한다. 이 휘처 모델은 휘처, 휘처 사이의 관계(feature relationship), 제약사항(constraints), 고려사항(issues and decisions)으로 이루어져 있다. 이때, 휘처들은 네 가지의 범주, 즉 "능력(capability)," "운용 환경(operating environment)," "영역 기술(domain technology)," 그리고 "구현 기술(implementation technology)" 휘처로 구분된다(그림 4 참조).

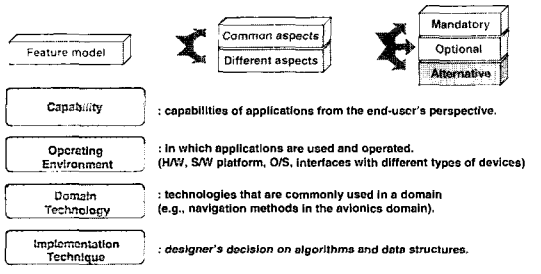


그림 4 휘처 모델의 개념

이러한 휘처들은 각각의 의미와 성격, 상호 연관성을 갖고 있으며, 이를 하나의 모델로 집약한 것이 휘처 모델이다. 휘처 자체의 성격으로는 mandatory, optional, alternative 휘처로 구분되는데, 이는 프로덕트 라인의 모든 프로덕트에서 반드시 존재해야만 하는 휘처, 선택이 가능한 휘처, 일련의 휘처 중에서 반드시 하나만 존재해야 하는 휘처를 각각 나타낸다. 휘처 사이의 관계로는 "composed-of," "generalization/specialization," "implemented-by"가 있으며, 이외에 휘처들 사이의 "require" 그리고 "mutually-exclude"와 같은 제약사항과, 휘처 선택의 지침을 제공하는 고려사항이 있다(휘처 모델링에 대한 프로세스와 가이드라인은 [28]에 기술되어 있다).

이렇게 어떤 프로덕트 라인을 분석하여 개발된 휘처 모델의 중요성과 의미는 다음과 같다.

- 프로덕트 라인에 속해 있는 프로덕트 사이의 공통점과 차이점을 분석하게 해 준다.
- 공통점으로 분석된 휘처는 재사용의 가능성이 높은 부분을 나타낸다.
- 차이점으로 분석된 휘처는 아키텍처 및 컴포넌트에서 변화를 수용해야 하는 부분을 나타낸다.
- 휘처들의 의미를 표준화하고 하나의 모델로 집약함으로써, 의사소통의 매체로 사용한다.
- 휘처 모델을 이용하여 프로덕트 라인의 영역(scope)을 명확히 한다.
- 프로덕트 휘처들 사이의 관계가 가시화된다.

휘처 모델링과 동시에 수행되는 활동으로 프로덕트 라인 요구사항 분석[29] 활동이 있으며, 이 활동의 결과로 유즈 케이스(use case)모델, 분석 객체(analysis object)모델, 그리고 품질 요소 등이 분석되고 개발된다. 이러한 프로덕트 라인 요구사항과 휘처모델은 MPP를 정련하는데 사용된다. 즉, MPP를 개발하는데 고려하지 않았던 사항인 프로덕트 운용 환경이나 구체적인 개발 방법 등이 MPP에 반영되며, 이렇게 정련된 MPP는 컴포넌트 설계에 필요한 제약사항과 고려사항을 보다 구체적으로 제공하게 된다.

3.3 아키텍처 및 컴포넌트 설계

본 절에서는 프로덕트 라인 자산 개발 프로세스에 포함되어 있는 네 가지 활동인 아키텍처 설계 및 정련 그리고 설계 객체 모델링 및 컴포넌트 설계에 대하여 설명한다(그림 2 참조). 이 단계에서는 앞에서 개발된 산출물인 MPP와 휘처 모델 그리고 프로덕트 라인 요구 사항을 이용하여 프로덕트 라인 아키텍처를 설계하고 그 구성요소들을 개발하게 된다.

FORM의 아키텍처 개발 프로세스는 개념적 아키텍처 설계와 아키텍처 정련으로 이루어져 있다. 개념적 아키텍처 설계는 시스템의 추상적, 상위 수준의 기능적 구성요소를 식별하고 여기에 휘처를 할당하며 구성요소들 사이의 데이터 및 제어 관계를 나타내

는 활동이다. 이러한 개념적 아키텍처는 프로세스(process) 아키텍처와 배치(deployment) 아키텍처로 정련된다. 프로세스 아키텍처는 기능적 요소들이 할당된 프로세스(즉, 프로세서 상에서 동시에 수행되는 단위)를 기술하여 시스템 내의 concurrency를 나타낸 아키텍처이며, 배치 아키텍처는 이러한 프로세스들이 실제로 어떤 하드웨어 자원(즉, 프로세서, 메모리 등)에 할당되는지를 나타내는 아키텍처이다. 개념적 아키텍처의 구성요소를 정련하여 프로세스 아키텍처로 할당하면서 특히 고려해야 할 사항으로는, 프로세스의 형태(예: 상주 프로세스 혹은 운용 중에 생성과 소멸을 반복하는 프로세스)와 프로세스의 수(예 : 단일 프로세스로 처리할 것인지 아니면 다수의 concurrent한 프로세스로 중복 할당하여 처리할 것인지), 그리고 각 프로세스 사이의 통신 방법[예 : 메시지 큐(message queue)방식, 응답을 기다리는 방식(message with reply) 등]이 있다.

한편, 휘처 모델, 프로덕트 라인 요구사항, 그리고 개념적 아키텍처를 기반으로 설계 객체 모델이 개발되게 된다. 이때 프로덕트 라인을 구현하는데 필요한 상용(commercial off the shelf : COTS) 컴포넌트나 디자인 패턴 등을 고려하여야 한다(휘처 모델 기반의 설계 객체 모델링 방법은 [25]에 기술되어 있다). 그리고, 컴포넌트 설계는 설계 객체 모델을 이용하여 프로세스 아키텍처와 배치 아키텍처를 기반으로 실제 시스템을 구성하게 될 컴포넌트를 개발하는 활동이다. 이러한 모든 활동의 결과로서 프로덕트 라인 아키텍처와 이를 구성하는 컴포넌트가 개발된다.

아키텍처 설계 및 정련 그리고 컴포넌트 설계 과정에서 반드시 고려해야 할 것은, MPP, 휘처모델, 그리고 프로덕트 라인 요구사항에서 식별된 품질 요구사항 및 제약사항을 만족하도록 하는 것이다. 예를 들면, 참조 아키텍처는 고객이 요구하는 성능 요구사항을 만족시켜야 하며, 이를 확인하기 위하여 아키텍처를 평가하는 것이 필요하다[30]. 다른 예로서, 마켓의 요구에 의해서 프로덕트에 모든 휘처를 미리 패키지화하여 납품하는 경우와 각 고객에 맞게 휘처를 선택하여 납품하는 경우가 있다면, 참조 아키텍처 및 컴포넌트는 이를 수용할 수 있도록 설계/구현되어야 한다.

4. 프로덕트 개발 프로세스

FORM의 프로덕트 개발 프로세스는 프로덕트 자산 개발 프로세스에서 개발된 자산을 이용하여 특정 프로덕트를 개발하는 활동으로 구성되어 있다. 즉, 프로덕트 라인에 속해있는 특정 프로덕트를 개발하는 프로세스이다. 이 프로세스는 우선 고객의 요구사항을 기반으로 프로덕트 요구사항을 분석하고 이에 적합한 휘처를 휘처모델에서 선택하는 활동, 그리고 참조 아키텍처를 선택 및 적응시키는 활동, 마지막으로 컴포넌트를 아키텍처에 적응시키고 코드를 생성하는 활동으로 구성되어 있다(그림 2의 하단 참조).

위에서 언급한 바와 같이, 프로덕트 개발 프로세스는 프로덕트의 요구사항을 분석하는 것으로 시작되는데, 이러한 활동은 휘처 모델에서 휘처를 선택하는 것과 동시에 이루어진다. 휘처 모델은 단순히 고객이 선택 가능한 휘처들만으로 이루어진 것이 아니라, 휘처들 사이의 다양한 관계를 미리 분석하여 모델화한 것이기 때문에, 고객이 휘처를 선택함에 따라서 이후에 선택할 수 있는 휘처의 폭이 점차로 줄어들게 된다. 예를 들면, 서로 "mutually exclude" 관계에 있는 휘처의 경우, 고객이 이들 중 하나의 휘처를 선택한다면, 다른 휘처는 선택할 수 없게 된다. 따라서 휘처 모델은 선택된 휘처의 집합이 시스템 구성의 유효성(validity)과 일관성(consistency)을 갖도록 유도해 준다. 또한, optional 휘처를 선택할 때에는 각 선택에 따른 장단점을 "고려사항" 으로 제공하게 되어 고객이 자신에게 필요한 휘처를 선택할 수 있도록 도와준다. 예를 들면, "사설 교환기의 ISDN 휘처는 음성과 데이터의 동시 서비스를 가능하게 해주나, 일반 음성 전화 휘처보다 더 많은 비용이 소요된다"는 등의 정보를 고객에게 제공하게 된다.

프로덕트 요구 분석과 휘처 선택이 완료되면, 이에 합당한 아키텍처 모델이 참조 아키텍처 중에서 선택된다. 이러한 아키텍처의 선택은 주로 품질요소에 의해서 결정되는 경우가 많은데, 예를 들면, 수십 명 내외의 소규모 사용자를 수용하는 사설 교환기와 수천 명의 사용자를 수용해야 하는 사설 교환기의 아키텍처는 그 성능 및 신뢰도에 대한 요구사항이 다르므로 결국 서로 다른 아키텍처 모델을 필요로 한다. 이러한 휘처 선택에 기반을 둔 아키텍처 모델이 결정된 후, 이 아키텍처에 컴포넌트를 적응시키고 조합해 넣는 과정이 수행된다.

컴포넌트가 요구사항과 아키텍처의 변화를 수용하는 방법의 하나로, FORM에서는 마크로 처리

(macro processing)[23] 기능을 제공한다. 즉, 컴포넌트 내에 프로덕트의 변화를 수용해야 할 부분은 마크로 처리를 위한 언어로 명세하게 되고, 이 부분은 삭제, 변경, 혹은 그대로 유지되어 최종 프로덕트에 반영된다. 예를 들면, 시스템 유지보수를 위한 사용자 인터페이스 메뉴 중에서 optional 훠처가 있는 경우, 이 훠처를 구현한 컴포넌트의 부분이나 관련 컴포넌트는 고객의 선택에 따라서 최종 프로덕트에 포함되거나 삭제될 수 있다. 다른 예로서, 고객에 따라 다른 값(value)이 할당될 수 있는 부분(예 : 전자 게시판의 최대 접속 허용 인원 등)은, 코드생성 시 고객이 지정한 값으로 자동 변경된다.

5. 결 론

프로덕트 라인 개발 방법은 프로덕트 라인 자산을 개발하고 이를 이용하여 프로덕트를 개발하고 진화시킴으로서, 생산성을 높이고 마켓 진입까지의 시간을 줄이는 가장 신뢰할 수 있는 방법 중에 하나이다. 이때, 재사용이 가능하도록 자산을 개발하기 위해서는, 우선 프로덕트들의 공통점과 차이점을 분석하고 이 결과물을 이용하여 자산을 개발하여야 한다.

FORM 방법은 다양한 산업체의 프로덕트 라인 프로젝트에 적용되었으며, 이러한 경험을 통하여 얻은 교훈은 다음과 같다:

- 하나의 프로덕트를 개발하는 것과 비교할 때, 프로덕트 라인 방법을 적용하는 것은 일반적으로 큰 규모의 초기 투자와 긴 개발 기간을 필요로 한다. 그러므로 프로덕트 라인 방법을 적용해 본 경험과 기술이 없는 조직이 최초로 이를 적용하려고 할 때는, 소규모의 프로덕트 라인, 혹은 프로덕트 라인의 일부에 대해서 우선 적용해 볼 것을 권장한다. 또한, 프로덕트 라인을 구성하고자 하는 영역의 전문가가 조직 내부에서 이 프로젝트에 반드시 참여할 수 있어야 한다.
- 프로덕트 개발 완료시기가 촉박한 프로젝트를 대상으로 프로덕트 라인을 구성하려고 하는 경우, 프로젝트가 진행됨에 따라 프로덕트 라인의 구성 보다는 납기를 맞추기 위하여 하나의 프로덕트 개발로 그 초점이 옮아가는 경향이 있다. 따라서 이러한 프로젝트는 적용 대상에서 제외하는 것이 좋다.
- 프로덕트가 운용되는 환경이 아주 상이하게 다른 경우, 다른 프로덕트 라인으로 식별하는 것이 바람직하다. 예를 들면, 엘리베이터 제어 소프트웨어의 경우, 화물용 엘리베이터와 승객용 엘리베이터가 인터페이스 하는 하드웨어 환경이 상이하여, 이들 인터페이스를 추상화하여 함께 쓸 수 있도록 설계하는 것이 어렵다. 이런 경우, 이들을 각각의 프로덕트 라인으로 취급하는 것이 유리하다.
- 표준화된 시스템 인프라 구조를 갖고 있는 조직의 경우, 표준에서 추후에 발생 가능한 변화를 제시했을 때, 이를 잘 수용하지 않으려는 경향이 있다. 표준화된 환경에서 작업을 하던 엔지니어들은, 표준이 결코 바뀌지 않으리라는 믿음을 갖고 있으며, 이는 프로덕트 라인 방법을 적용하는데 걸림돌이 된다. 그러므로 기술이 발전함에 따라서 표준도 결국 변화할 것이라는 것을 설득하고 이러한 변화를 수용하기 위한 설계와 개발에 참여하도록 유도하는 것이 중요하다.

프로덕트 라인 개발은 대규모의 초기 투자를 필요로 하며, 때로 조직의 구조적, 문화적 변화를 요구하기도 한다. 따라서 프로덕트 라인 방법을 성공적으로 적용하기 위해서는 강력하고 장기적인 경영진의 의지가 반드시 필요하다.

참고문헌

[1] J. Neighbors, The Draco Approach to Construction Software from Reusable Components, *IEEE Transactions on Software Engineering*, SE-10(5), 564-573, September 1984.

[2] K. Kang, S. Cohen, J. Hess, W. Nowak, and S. Peterson, Feature-Oriented Domain Analysis (FODA) Feasibility Study, *Technical Report CMU/SEI-90-TR-21*, Pittsburgh, PA, Software Engineering Institute, Carnegie Mellon University, November 1990.

[3] R. Prieto-Diaz, Implementing Faceted Classification for Software Reuse, *Communications of the ACM*, 34(5), 88-97, May 1991.

[4] S. Bailin, Domain Analysis with KAPTUR, *Tutorials of TRI-Ada'93*, I, ACM, New York, NY, September 1993.

[5] Software Productivity Consortium, Reuse-Driven Software Processes: Guidebook Version 02.00.03, *SPC-92019-CMC*, Herndon, VA, Software Productivity Consortium, 1993.

[6] M. Simos et al., Software Technology for

Adaptable Reliable Systems (STARS) Organization Domain Modeling (ODM) Guidebook Version 2.0, *STARS-VC A025/001/00*, Manassas, VA, Lockheed Martin Tactical Defense Systems, 1996.

[7] J. Coplien, D. Hoffman, and D. Weiss, Commonality and Variability in Software Engineering, *IEEE Software*, 15(6), 37-45, November/December 1998.

[8] K. Schmid, Scoping Software Product Lines, Proceedings of the 1st Software Product Line Conference (SPLC), August 28-31, 2000, Denver, Colorado, USA, Patrick Donohoe (Ed.), *Software Product Lines: Experience and Research Directions*, 3-22, Kluwer Academic Publishers, 2000.

[9] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, and M. Stal, *Pattern Oriented Software Architecture: A System of Patterns*, Chichester, England, John Wiley & Sons Ltd., 1996.

[10] K. Schmid and C. Gacek, Implementation Issues in Product Line Scoping, Proceedings of the 6th International Conference on Software Reuse (ICSR), Vienna, Austria, June 2000. W. Frakes (Ed.), *Software Reuse: Advances in Software Reusability*, New York, NY: Springer Verlag, June 2000.

[11] D. M. Weiss and C. T. R. Lai, Software Product Line Engineering: A Family Based Software Development Process, Reading, MA: Addison Wesley Longman, Inc., 1999.

[12] J. Bosch, Design and Use of Software Architectures: Adopting and Evolving a Product-line Approach, Reading, MA: Addison Wesley Longman, Inc., 2000.

[13] P. Donohoe, (Ed.) *Software Product Lines: Experience and Research Directions*, Kluwer Academic Publishers, 2000.

[14] P. Clements and L. Northrop, *Software Product Lines: Practices and Patterns*, Boston, MA: Addison Wesley Longman, Inc., 2001.

[15] A. D. Vici and N. Argentieri, FODAcom: An Experience with Domain Analysis in the Italian Telecom Industry, Proceedings of the 5th International Conference on Software Reuse (ICSR), Victoria, BC, Canada, 166-175, June 1998.

[16] N. S. Zalman, Making the Method Fit: An Industrial Experience in Adopting FODA, Proceedings of the 4th International Conference on Software Reuse (ICSR), Orlando, FL, 233-235, 1996.

[17] M. L. Griss, J. Favaro and M. d'Alessandro, Integrating Feature Modeling with the RSEB, Proceedings of the 5th International Conference on Software Reuse (I, Victoria, BC, Canada, 76-85, June 1998.

[18] M. L. Griss, Implementing Product-Line Features by Composing Aspects, Proceedings of the 1st Software Product Line Conference (SPLC), August 28-31, 2000, Denver, Colorado, USA, Patrick Donohoe (Eds.), *Software Product Lines: Experience and Research Directions*, 47-70, Kluwer Academic Publishers, 2000.

[19] A. Hein, M. Schlick, and R. Vinga Martins, Applying Feature Models in Industrial Settings, Proceedings of the 1st Software Product Line Conference (SPLC), August 28-31, 2000, Denver, Colorado, USA, Patrick Donohoe (Eds.), *Software Product Lines: Experience and Research Directions*, 47-70, Kluwer Academic Publishers, 2000.

[20] S. G. Cohen, J. L. Stanley Jr., A. S. Peterson, and R. W. Krut Jr., Application of Feature-Oriented Domain Analysis to the Army Movement Control Domain, *Technical Report, CMU/SEI-91-TR-28*, ADA 256590, Pittsburgh, PA, Software Engineering Institute, Carnegie Mellon University, 1991.

[21] R. Krut and N. Zalman, Domain Analysis Workshop Report for the Automated Prompt Response System Domain, *Special Report, CMU/SEI 96 SR 001*, Pittsburgh, PA, Software Engineering Institute, Carnegie Mellon University, 1996.

[22] K. Czarnecki and U. Eisenecker, *Generative Programming: Methods, Tools, and Applications*, Addison-Wesley, New York, 2000.

[23] K. Kang, S. Kim, J. Lee, K. Kim, E. Shin and M. Huh, FORM: A Feature Oriented Reuse Method with Domain Specific Reference Architectures, *Annals of Software Engineering*, Vol.5, 143-168, 1998.

[24] K. Kang, S. Kim, J. Lee, and K. Lee, Feature-Oriented Engineering of PBX Software for Adaptability and Reusability, *Software Practice and Experience*, 29(10), 875-896, 1999.

[25] K. Lee, K. Kang, W. Chae, and B. Choi, Feature-Based Approach to Object-Oriented Engineering of Applications for Reuse, *Software-Practice and Experience*, 30(9), 1025-1046, 2000.

[26] K. Lee, K. Kang, E. Koh, W. Chae, B. Kim, and B. Choi, Domain-Oriented Engineering of Elevator Control Software: A Product Line Practice, Proceedings of the 1st Software Product Line Conference (SPLC), August 28-31, 2000, Denver, Colorado, USA, Patrick Donohoe (Eds.), *Software Product Lines: Experience and Research Directions*, 3-22, Kluwer Academic Publishers, 2000.

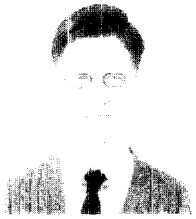
[27] K. Kang, F. Bachmann, L. Bass, and P. Donohoe, Product Line Component Design: Marketing and Product Plan as a Key Design Driver, *Technical Report*, Pittsburgh, PA, Software Engineering Institute, Carnegie Mellon University (in progress).

[28] K. Lee, K. Kang, and J. Lee, Concepts and Guidelines of Feature Modeling for Product Line Software Engineering, Accepted for publication at the 7th International Conference on Software Reuse (ICSR), Austin, Texas, USA, April 15-19, 2002.

[29] G. Chastek, P. Donohoe, K. Kang, and S. Thiel, Product Line Analysis: A Practical Introduction, *Technical Report CMU/SEI-2001-TR-001*, Pittsburgh, PA, Software Engineering Institute, Carnegie Mellon University, 2001.

[30] L. Bass, P. Clements, and R. Kazman, *Software Architecture in Practice*, Reading, MA: Addison Wesley Longman, Inc., 1998.

이 재 준



1991 서강대학교 수학과 졸업(이학사)
1998 포항공과대학교 정보통신대학원 졸업(공학석사)
1993~2000 LG 정보통신 중앙연구소 주임 연구원
2000~2001 포항공과대학교 정보통신 연구소 전임연구원
2001~현재 포항공과대학교 컴퓨터공학과 박사과정
관심분야:Software product line engineering, software reuse, software architecture
E-mail:gibman@postech.ac.kr

강 교 철

1973 고려대학교 계산통계학과 졸업(이학사)
1976 University of Colorado(공학석사)
1982 University of Michigan(공학박사)
1982~1984 University of Michigan, Visiting Professor
1984~1985 Bell Communications Research, Technical Staff
1985~1987 AT&T Bell Labs., Technical Staff
1987~1992 SEI, Carnegie Mellon University, Senior Member, Project Leader
1992~현재 포항공과대학교 컴퓨터공학과, 교수
관심분야:Software product line engineering, software reuse, software architecture, software specification
E-mail:kck@postech.ac.kr