재사용 기반의 소프트웨어 개발 체계 구축 : 내장형 소프트웨어 영역의 기업 사례

김 강 태†

요 약

본 논문은 CE(Consumer Electronics) 제품을 개발하는 기업에서 소프트웨어 재사용의 향상을 위하여 기업 내에서 추진한 여러 사례들을 기반으로 소프트웨어 재사용에 대한 기반을 구축하고, 이를 개발 조직에 적용하면서 발생한 문제점의 도출과 개선점의 반영에 대해 논한다. 기업 내에서의 재사용 기반을 구축하기 위해서는 기술적, 관리적, 개발환경의 측면이 모두 고려된 종합적인 접근 방법이 필요하다. 본 논문에서는 기술적 측면에서 개발 방법론을, 관리적 측면에서 재사용 메트릭을, 그리고 개발환경의 측면에서 재사용 저장소를 개발하고 운영한 사례를 논하며, 각각을 적용하면서 발생한 문제점을 분석하여 기업에서 보다 효과적인 재사용 기반을 구축하는 개선사례를 제시한다.

본 논문에서 다루는 재사용 활동의 대상은 개발대상 측면에서 내장 소프트웨어라는 특성, 조직적 측면에서 다양한 제품을 다루는 수십 개의 각기 다른 개발영역과 조직구조를 가진 대규모 조직이라는 특성, 그리고 제품 개발 주기가 매우 짧으며 동일 제품에 대한 파생 제품이 동시에 다량으로 개발된다는 개발 환경의 특성에 기반하고 있다. 본 논문은 상기 분야에 대한 소프트웨어 재사용 현황에 대한 보고서로, 또한 이로부터 기업 내 재사용 기반을 개선하는 사례와 그 방안을 제시한다는 측면에서의 활용도가 있겠다.

키워드 : 재사용 전략, 재사용 방법, 재사용 프로세스, 재사용 메트릭, 재사용 저장소

Enabling reuse driven software development : lessons learned from embedded software industry practice

Kangtae Kim†

ABSTRACT

This paper presents industry feedback and a case of improvement trial on enabling reuse driven software development which is one of several activities to improve software quality and productivity in a company which develops software that are embedded into consumer electronic products. Several case studies will be introduced that are related to software reuse strategies and practices to show how to establish environment for reuse basis in a company, how to apply it to development team and project and how to improve that through trials and errors. To enable reuse-oriented software development in a huge company, integrated and focused approach is needed among technical, management and environmental point of view. We tried to solve that problem in technical field with reuse method, in management filed with reuse metric and in environment field with reuse repository.

The characteristics of our software development environment could be summarized as below. The first, embedded software which would not independent to hardware devices and the second, it is very huge company which develops extremely various products by many different organization with different domain characteristics and the third, development lead time is extremely short and many variation models are stems from basic models. We expect that our study would give contribution to industry struggling to solve similar problem for presenting our experience and could be a reference model for enabling software reuse in a real world practically.

Key Words : Reuse Strategy, Reuse Method, Reuse Process, Reuse Metric, Reuse Repository

1. 서 론

전자제품에서 소프트웨어의 비중은 날로 높아가고 있다. 새로운 IT(Information Technology)기술이 전자제품에 적용되고 있으며, 사용자들 역시 기존의 전자제품에서 얻을 수 있는 기능을 초월하는 수많은 다양한 기능을 요구하고 있다. 이는 결국 전자제품 내에서 소프트웨어가 차지하는 비중이 점점 커지고 있다는 것을 의미하며, 전자제품 내에 내장되는 소프트웨어의 복잡도와 크기도 증가한다는 것을 의미한다. IT기술은 제품의 네트워크화, 퍼스널화, 모바일화를 가

† 정 회 원 : 삼성전자 책임연구원
논문접수 : 2006년 1월 19일, 심사완료 : 2006년 3월 20일

능케 하였고, 고객은 이러한 기술을 바탕으로 기존의 제품에서 제공하지 못했던 수준 높은 서비스를 요구하고 있다. 기존의 단품을 구동시키는 단순 내장형 소프트웨어에서 제품간 네트워킹 및 커넥티비티를 확대시킬 수 있는 네트웍 소프트웨어, 차별화 기능을 부여하는 어플리케이션 소프트웨어, 더 나아가 소프트웨어 서비스의 영역까지 확대되고 있다. 따라서 점점 복잡해지고 커져가는 내장 소프트웨어를 효율적으로 개발하는 것은 전자업계가 풀어야 할 중요한 숙제라 할 수 있다[1, 2].

이를 위해 본 방법론이 적용된 기업에서도 소프트웨어를 체계적으로 개발하기 위한 소프트웨어 개발 프로세스를 제품개발 프로세스 내에 정의하고 관리하는 프로세스적 접근방법, 소프트웨어 재사용 자산을 개발하고 이를 활용하는 재사용 기법, 개발자의 생산성 향상에 필요한 자동화된 도구의 개발 및 적용 등을 병행 진행하고 있다[3]. 본 논문은 이러한 접근 방법 중에서 재사용이 가능한 소프트웨어 자산을 생산, 관리, 사용하여 개발의 효율성을 향상시키고, 기 검증된 자산의 재사용을 통한 품질 향상을 목적으로 하는 소프트웨어 재사용을 위해 기술적, 관리적, 환경적 측면에서 기업 내에서 추진한 사례를 통해 보다 효과적인 재사용을 위한 방안을 제시한다.

본 논문은 내장형 소프트웨어 영역이면서 동시에 다품종의 대규모 제품 생산을 하는 대규모의 기업에서 소프트웨어 재사용을 위해 필요한 기술, 환경 및 도구에 대한 종합적인 기업활동을 사례로 참고할 수 있겠으며, 조직적 차원의 재사용 전략 및 기법에 대한 기업체의 피트백으로 볼 수 있을 것이다.

본 논문의 구성은 다음과 같다. 2장에서는 논문의 배경 및 주요 관련연구를 정리하였고, 3장에서는 재사용 기반 구축의 사례로 재사용 방법론, 재사용 메트릭, 재사용 저장소 각각에 대한 구축 및 적용 사례를 논하였으며, 4장은 결론으로 적용 사례를 통해서 얻어낸 문제점 및 교훈을 분석하고 이를 바탕으로 각 사례에 대한 개선방안을 도출한다.

2. 배경 및 관련연구

2.1 소프트웨어 재사용

재사용의 목적은 소프트웨어 품질과 생산성을 향상시키는 것이다. 점점 커지고 복잡해지는 소프트웨어를 보다 빨리, 보다 품질 높게 개발하기 위해서 재사용은 필수적이다. 소프트웨어 재사용은 기존의 소프트웨어 혹은 소프트웨어 개발과 관련된 기술적 지식을 재활용하여 새로운 소프트웨어를 개발하는 행위이다[4]. 재사용의 대상을 일반적으로 재사용 자산(reuse asset)이라 하며, 소프트웨어 개발과 관련된 모든 재사용 가능한 단위로 정의할 수 있다. 객체지향의 개념의 대두, 재사용 컴포넌트를 기반으로 소프트웨어를 개발하는 컴포넌트 기반 개발 방법(CBD : Component based development), 설계 지식을 재사용하는 디자인 패턴, 소프트웨어의 상위설계 및 기본 구조를 재사용 대상으로 하는 소프트웨어 아키텍처, 제품군에 대응할 수 있는 소프트웨어 프로덕트 라인(Software product line)까지 재사용의 대상은 점점 그 크기와 적용대상이 넓어져 가고 있다[5, 6].

본 논문에서 제시되는 방법론은 CBD를 기반으로 프로덕트 라인 공학 기법을 반영하였다.

컴포넌트란 물리적으로 대체 가능한 소프트웨어 시스템의 구성요소로 인터페이스의 집합을 제공하며, 특정 단위로 패키징되어 기능 및 서비스를 미리 정의된 인터페이스를 통해 제공하는 단위이다. 컴포넌트 기반 개발 방법은 이러한 컴포넌트를 기반으로 소프트웨어를 개발하는 방법으로 재사용 단위인 컴포넌트를 개발하기 위한 컴포넌트 개발 단계로 For reuse, 기 존재하는 컴포넌트를 조합하여 소프트웨어 시스템을 개발하는 With reuse로 나눌 수 있다[7]. 재사용 대상이 되는 컴포넌트 자체를 개발, 재사용하는 데에 초점이 맞추어져 있는 컴포넌트 기반 개발 방법은 기 검증된 효과적인 방법이다[8].

최근에는 대부분의 소프트웨어 시스템이 처음부터 개발되는 것이 아니라 기존의 시스템에서 변경, 파생되어 개발되는 소프트웨어 프로덕트 라인 방법이 대두되고 있다. 개발조직에서 신규로 개발되는 소프트웨어는 그 조직의 비즈니스 영역을 넘나들지 않고 그 영역에 한정적이며 이에 따라서 기존에 개발하는 소프트웨어에서 크게 벗어나지 않는 개발범위를 가지고 있다는 것이다. 이러한 가정 하에 비즈니스의 영역 내에서의 재사용은 영역지식을 신중하게 분석하여 고정부와 변동부를 미리 정의하여 제품군에 대응할 수 있는 개발체계를 구축하여 보다 효과적인 재사용 기반의 개발이 될 수 있다는 것이다[8, 9].

2.2 재사용 저장소

컴포넌트 기반 개발 방법은 소프트웨어 개발 생산성을 향상시키는 검증된 방법으로 내장형 소프트웨어 영역에서도 효과적인 방법이다. 하지만 기업 조직 내에서 실질적인 재사용이 이루어지기 위해서는 단위요소인 컴포넌트를 관리하고 검색하기 위한 저장소의 개발이 필요하다[10].

1990년 대 초 컴포넌트를 추출하기 위한 기법들이 제안되었으며, 주로 input/output 패러미터에 의한 문법 기반(Syntax based)의 기법이었다[11, 12]. 이는 컴포넌트의 정확한 행위(Behavior)를 표현할 수 없는 한계를 가지고 있었으며 이러한 문제에 대한 해결책으로 의미 기반(Semantic based) 추출 기법들이 제안되었다. 이러한 기법들에서 컴포넌트의 행위는 형식 언어(Formal specification language)에 의해 기술된다[13]. 또한 복잡한 형식언어 사용에 대한 개선된 기법으로 룰 기반 해석 기법(Rule based reasoning) 등이 제안되어 컴포넌트 행위에 대한 추론 및 유사성 검증 등의 기술이 개선되었다[14]. 또한 공통어휘 및 분류체계 등의 온톨로지(Ontology) 기법을 적용한 개선된 기법들이 제안되었다[10].

내장형 소프트웨어 영역에서는 일반 애플리케이션 기반의 컴포넌트 저장소에서 보다 더 고려해야 할 해당 영역의 특수한 사항들이 있으며 대표적으로 메모리 제약, 응답 시간

제약 등의 컴포넌트의 비기능적 요소(NFR : Non Functional Requirement)에 대한 것이다[15, 16].

2.3 재사용 메트릭

재사용 메트릭은 소프트웨어 제품 및 프로세스에 대해 재사용 측면에서 수치적으로 측정할 수 있는 속성을 갖는 것으로 정의할 수 있다[17]. 소프트웨어 재사용은 공학적 측면에서 기본적으로 수치적인 측정과 분석이 이루어져야 하며, 실험적인 접근이 필요하다[18]. 재사용 메트릭은 지금까지 많은 기존 연구를 통해서 정의되어 왔다. 소프트웨어 재사용 메트릭에 대한 기존의 주요 연구들은 재사용 컴포넌트와 소프트웨어 개발의 주요 요소인 품질, 생산성과의 연관관계를 수립하기 위한 모델 및 측정치 개발과 그에 대한 실험이 순환적으로 진행되었다. 재사용의 형태에 대한 분류, 재사용 성숙도, 재사용 양에 대한 측정, 재사용 라이브러리에 대한 측정, 비용 대비 효과 측정, 재사용에 대한 심사모델 등이 이에 해당한다[19]. 개발 생산성, 개발 비용, 개발 기간 등의 소프트웨어 개발의 주요 변수에 대한 재사용의 효과들과 그에 대한 적용사례는 많은 연구들을 통해서 그 효과가 입증되고 있으나[20], 특정 조직에서 활용하기 위해서는 해당 조직에서 재사용에 대한 측정 모델의 개발과 실험을 반복하여 그 조직에 맞는 모델을 개발해야 한다[21].

3. 재사용 체계 구축 사례

관련 연구를 통해서도 알 수 있듯이 조직에서의 소프트웨어 재사용을 활성화하기 위해서는 그 기반환경 제공되어야 한다. 본 절에서는 기업 내 기반환경으로 재사용을 수행하기 위한 세부 활동과 관련 기술을 정의한 재사용 방법론, 재사용 자산의 공유를 위한 재사용 저장소 그리고 조직의 재사용 현황을 파악하고 효과를 분석하기 위한 재사용 지표를 개발하여 운영한 사례에 대해 논한다.

3.1 CBD 기반의 편집설계 방법론 개발 및 적용

전사 차원의 소프트웨어 재사용을 추진하기 위한 기반으로 재사용 가능한 소프트웨어 컴포넌트를 개발하고, 개발된 재사용 컴포넌트를 기반으로 소프트웨어를 개발하며, 재사용 컴포넌트를 등록/관리하기 위한 일련의 절차 및 관련 기법으로 "편집설계 방법론"을 개발하였다. 편집 설계 프로세스는 기본적으로 일반적인 컴포넌트 기반 개발 방법을 참조하여 개발환경에 맞게 조정하여 개발되었다. 편집설계 방법론의 개발 목적은 소프트웨어 재사용을 통한 개발 비용 감소, 납기 단축 및 품질 증대이다. 전자업체의 소프트웨어 개발은 주로 선행개발을 통한 기반 코드(Base code)를 개발하고 제품 파생 시 해당 제품의 하드웨어 환경에 맞도록 포팅하거나 일부 기능을 추가/수정하는 형태가 대부분이다. 따라서 재사용의 대상도 기반 코드에서 미들웨어 상단의 상위 레이어의 대부분은 그대로 재사용 가능하나 하드웨어와 관련된 하위 레이어는 빈번한 변경으로 재사용이 미비한 실정이다. 또한 여러 제품들이 동시 다발적으로 병행 개발되기 때문에 기반코드가 복수 파생하여 기반코드와 파생코드가 모두 제대로 관리되지 못하는 문제점이 발생한다. 편집설계 방법론은 이러한 개발 환경에서 상위 레이어는 컴포넌트 기반 개발 방법의 접근 방법을 따르고, 하위 레이어는 HAL(Hardware Abstraction Layer)이나 OSAL(O/S Abstraction Layer)과 같은 추상화 계층을 개발하는 형태로 접근하였다.

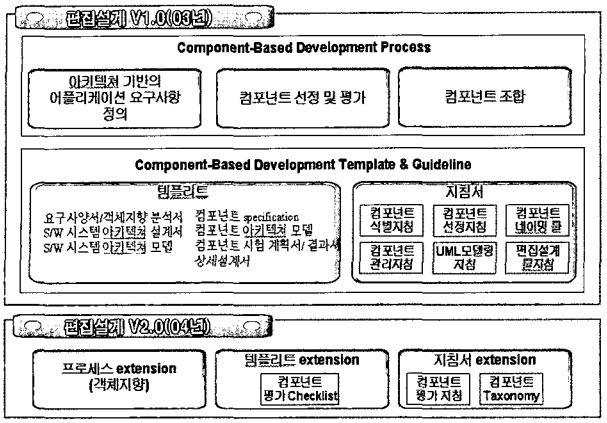
편집설계 방법론은 편집설계 프로세스, 편집설계 템플리트, 편집설계 지침으로 구성되며, 편집설계 프로세스는 제품 개발 프로세스 및 소프트웨어 개발 프로세스를 기준으로 요구분석에서 설계까지의 작업 절차를 정의한 것이다. 편집설계를 위한 컴포넌트 개발은 두 가지 방향으로 접근할 수 있다. 하나는 편집 설계에 필요한 요구 사항을 수집하여 편집 설계를 위한 재사용 자산으로서의 컴포넌트를 개발하는 것으로 컴포넌트 개발을 별도 과제화하는 것이며, 다른 하나는 과제 진행 시 과제의 산출물이 다른 과제에서 편집 설계가 가능하도록 소프트웨어 시스템을 컴포넌트화하고 후보 컴포넌트를 이용하여 새로운 컴포넌트를 개발해 내는 것이다. 편집 설계를 위한 컴포넌트 개발 지침은 기존의 개발 프로세스와 완전히 다른 것이 아니며, 기존의 개발 프로세스에 다음의 활동들이 추가적으로 고려되고 지원되어서 유기적으로 진행되도록 개발되었다.

- 컴포넌트를 사용하려는 각기 다른 사용자들의 각기 다른 요구 사항들을 취합하고 분석하여 각각의 요구 사항 간의 차이점을 파악하는 활동
- 분석된 요구 사항들의 차이점으로부터 구현 비용과 구현되었을 경우 얻을 수 있는 이익 등을 분석하는 활동
- 가능한 많은 개발자들이 편집 설계 활동에 반영할 수 있도록 적절한 수준의 일반성을 가지는 컴포넌트를 설계하는 활동

컴포넌트를 이용한 편집 설계 개발은, 설계 시에 이미 개발된 컴포넌트를 고려하여 편집 설계가 가능한 컴포넌트를 도출하고, 구현 시에 새로운 기능을 추가하고 이미 개발된 컴포넌트와 연계되어 동작할 수 있도록 편집하는 과정을 말하며, 컴포넌트를 편집 설계하여 새로운 소프트웨어 시스템을 구현할 경우 다음의 활동이 수행되도록 정의한다.

- 편집 설계 시 시스템의 요구 사항을 만족하는 후보 컴포넌트를 검색하는 활동
- 검색된 후보 컴포넌트로부터 시스템의 요구 사항을 가장 잘 만족하는 컴포넌트를 찾아 내기 위한 평가 활동
- 필요할 경우 특정 요구 사항을 만족시키기 위해 편집 설계를 위하여 기 개발된 컴포넌트를 변경시키는 활동

또한, 개발조직에서 편집설계가 적용되기 위해 편집설계를 수행하기 위해 필요한 조직 구성 및 개발 관련자들의 역할을 정의하였다. 편집설계 템플리트는 각 단계에서 정의되



(그림 1) 편집설계 방법론 구성

는 산출물을 하나의 문서로 패키징하였으며, 편집설계 지침은 컴포넌트를 식별하기 위한 컴포넌트 식별 지침, 기반 코드를 재공학(Re-engineering)하여 컴포넌트화의 대상을 선정하는 컴포넌트 선정 지침, 개발된 컴포넌트의 관리를 위한 컴포넌트 네이밍 룰과 컴포넌트 관리 지침, 모델링을 위한 UML 모델링 지침 그리고 편집설계 툴 지침으로 구성된다. (그림 1)은 편집설계 프로세스의 구성요소들을 보인다.

편집설계 방법론은 시범 조직을 선정하여, 중점적으로 시범과제를 추진하면서, 병행하여 편집설계 프로세스 정립을 추진하였다. 전사의 편집설계 추진 조직에서는 각 시범조직의 시범과제에 직접 기술지원을 수행하였고, 이를 우수사례로 전사에 확대 전파하는 형태로 적용되었다. 또한 소스코드의 재사용 차원을 넘어 모델링 정보의 재사용을 위해 모델링 언어인 UML의 도입과 이를 지원하는 설계 도구의 표준화를 진행하였다. 이를 통해서 전사 표준 설계 도구가 선정되고 이의 확산활동이 진행되었다.

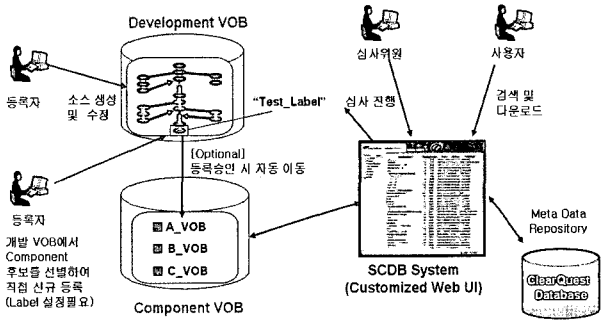
3.2 컴포넌트 저장소의 개발 및 운영

편집설계 방법론이 적용되면서 각 개발조직에서는 재사용 자산인 컴포넌트가 생산되기 시작하였다. 생산된 컴포넌트를 관리하고 이를 기반으로 소프트웨어를 개발하기 위한 기반 환경이 필요하였고 이를 위해 재사용 컴포넌트 저장소인 SCDB(Software Component DataBase)를 개발하였다. SCDB는 기본적으로 재사용 자산인 컴포넌트 및 그 관련 산출물의 저장소이며, 컴포넌트의 등록, 사용, 심사, 변경, 폐기를 위한 워크 플로우를 갖는 프로세스 도구의 기능을 포함한다.

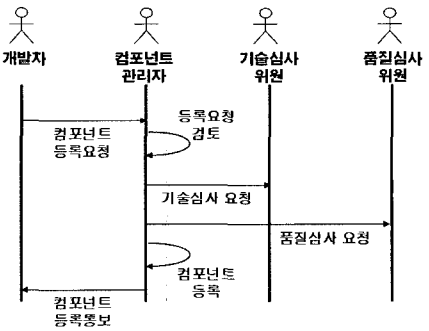
재사용을 지원하기 위한 시스템은 기본적으로 검색 기능, 재사용 자산의 버전 관리 기능 등이 필수적이다. 이러한 요구사항에 부합하기 위해 온톨로지를 이용하여 컴포넌트를 신속하게 찾기 위한 검색 기능을 제공하여 주제어, 분류체계에 대한 관심어 등을 통해 컴포넌트를 검색할 수 있게 한다. 특히 분류체계는 편집설계 방법론에 포함된 컴포넌트 분류체계 및 taxonomy를 그대로 반영한다. 재사용 자산의 버전관리를 위해 컴포넌트 별 버전 변경 내역과 관련 정보를 관리하는 기능을 제공한다. 변경 정보는 사용자의 새로운 변경 요구 등록, 처리, 및 재등록까지의 상태를 관리하며, 변경 이력과 변경 요청자, 변경 담당자 등의 정보를 제공한다.

(그림 2)에서와 같이 개발자는 개발과제 진행 중인 개발환경에서 후보 컴포넌트를 식별한 후 컴포넌트 분류체계에 의해 컴포넌트를 등록하는 것으로 재사용 프로세스가 시작되며, 별도의 심사 단계를 통해 검증된 컴포넌트는 정식 재사용 자산으로 관리된다. 등록 프로세스는 (그림 3)과 같이 관련자 간의 업무흐름으로 정의되어 있으며 기술심사는 컴포넌트의 재사용성에 대한 검증을, 품질심사에서는 기능적 검증이 수행된다.

초기 버전의 SCDB는 개발 저장소와 컴포넌트 저장소가 물리적으로 독립적으로 구현되어 형상관리 시스템과의 연계가 불가능하였다. 결국 개발자가 사용하는 형상관리 시스템의 저장소에 재사용 저장소의 컴포넌트를 다운로드 받는 형태로 사용하게 되어 컴포넌트 자체의 관리와 재사용된 컴포넌트의 관리가 분리되는 문제점이 있었다. 이후 SCDB의 물리적 저장소는 형상관리 시스템의 저장소를 그대로 사용하여 연계문제를 해결하였다. 결국 표준형상관리 시스템과 재사용 저장소가 연계되어 컴포넌트 저장을 위한 전용 저장소를 별도 운영하게 되었다. 별도의 저장소 운영으로 개발과제의 저장소로 활용되는 저장소에서 컴포넌트를 식별하게 되어 운영 상의 효율성을 높일 수 있었다.



(그림 2) SCDB의 구성도



(그림 3) 컴포넌트 등록 프로세스

3.3 재사용 지표 운영

표준화된 재사용 지표 관리를 통해 전사의 소프트웨어 재사용 현황을 파악하고, 재사용 업무의 활동을 가시화하며,

〈표 1〉 표준 재사용 메트릭

구분	지표
함수 레벨의 재사용	함수 재사용율(Reuse Ratio of Functions)
	함수 공용화율(Creation Ratio of Reusable Function)
컴포넌트 레벨의 재사용	컴포넌트 재사용율(Reuse Ratio of Components)
	컴포넌트 공용화율 (Creation Ratio of Reusable Components)
SCDB운영	전체 등록 수
	월별 등록 수
	월별 사용 수
	월별 변경 수

새로운 재사용 기술의 도입 효과 파악하기 위해 재사용 지표를 수립하고 운영하였다.

재사용 지표는 <표 1>과 같이 3가지 종류로 나누어 운영하였다. 소프트웨어 재사용 지표는 함수 레벨, 컴포넌트 레벨의 재사용이다. 함수 재사용율은 소스 코드 수준에서의 재사용 정도를 나타내는 지표로 전체 소프트웨어 중에서 소스코드의 재사용된 함수의 크기 비율을 나타낸다. 이를 위해서 수집해야 할 데이터는 전체 LOC(Lines of code)와 재사용된 함수의 LOC이다. 이때 함수의 변경 정도는 고려되지 않는다. 즉, 함수의 변경에 대한 난이도 및 투입되는 시간 자원 등에 대한 고려는 배제된다는 것이다. 함수 레벨의 공용화 지표인 함수 공용화율은 새로 개발한 소프트웨어 중에서 향후 재사용될 수 있는 함수의 크기 비율로 재사용 가능한 소프트웨어 단위(함수)의 개발 정도를 측정하는 지표이다. 이를 위해서 수집해야 할 데이터는 과제에서 신규 개발 LOC와 재사용될 수 있는 함수의 LOC로 신규개발 LOC에는 재사용된 코드는 제외된다. 컴포넌트 레벨의 재사용은 컴포넌트 재사용에 관련된 지표로 그 세부 내용은 함수 레벨의 재사용 지표와 동일하나 그 대상을 SCDB에 공식 자산으로 저장된 재사용 컴포넌트로 한정한다. 재사용 컴포넌트는 컴포넌트 기준을 만족해야 한다. 컴포넌트에 대한 기준은 1. 공개된 인터페이스를 통해서 사용 가능하도록 설계된 모듈이며, 2. 컴포넌트의 기능/동작/구조를 설명하는 컴포넌트 명세서와 함께 패키지화 되어 제공되어야 하며, 3. S/W 컴포넌트의 품질을 보증할 수 있는 시험 문서가 패키지에 포함되어 있어야 한다. 각 지표에 대한 계산식은 (그림 4)와 같다.

SCDB 운영 지표는 SCDB에 등록 및 사용되는 컴포넌트의 개수를 파악하는 현황파악 지표로 SCBD에 등록된 컴포넌트 전체 개수와 신규 등록, 변경, 폐기 등에 의한 주기적인 변경사항을 보이는 지표로 구성되었다. 재사용 지표를 운영하기 위해서 재사용 지표 수집 지침서와 템플리트를 개발하여 개발조직에서 활용할 수 있도록 하였으며, LOC를 측정하기 위한 LOC 측정 도구를 자체 개발하였다.

표준 재사용 지표는 개발 조직 별로 기반 코드를 기준으로 파생 코드에 대한 함수 레벨 및 컴포넌트 레벨의 재사용 지표를 수집, 분석하여 (그림 5)와 같이 제품군별로 재사용율에 대한 정보와 제품 내 특정 부분에 대한 재사용율에 대한

(1) 함수 재사용율

$$RRF\% = \frac{\sum \text{LOC of Reused Functions}}{\text{Total LOC}} \times 100$$

(2) 함수 공용화율

$$CRRF\% = \frac{\sum \text{LOC of Reusable Function}}{\text{Developed LOC}} \times 100$$

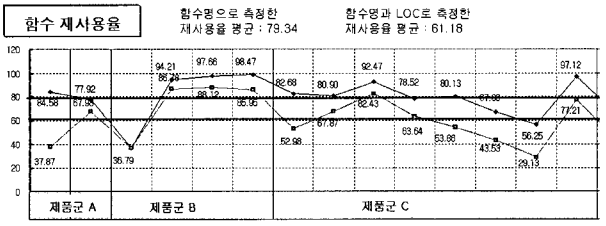
(3) 컴포넌트 재사용율

$$RRC\% = \frac{\sum \text{LOC of Reused Component}}{\text{Total LOC}} \times 100$$

(4) 컴포넌트 공용화율

$$IPCR\% = \frac{\sum \text{LOC of Reusable Component}}{\text{Developed LOC}} \times 100$$

(그림 4) 재사용 지표



(그림 5) 함수 재사용율 수집 사례

정보를 제공한다. 실제 연 2회에 걸쳐 개발 조직 별로 함수명으로 측정한 재사용율은 매우 높은 수준이었으며 이는 동일 제품군 내에서 소프트웨어의 기능은 안정적이어서 전체 소스 코드의 차원에서는 변경량이 매우 적은 것으로 분석되었다.

특정 개발조직을 대상으로 재사용 지표를 수집한 예로 (그림 5)에서 보듯 소스레벨의 재사용율은 평균적으로 79%에 이름을 알 수 있었다. 조직 별로 함수 수준으로 재사용이 활발하였고 함수명은 동일하나 그 내용을 수정하여 재사용하는 함수 비중이 높았다. 또한 함수 레벨의 지표는 모든 개발조직에서 산출 가능하였으나 컴포넌트 레벨의 지표는 일부 조직에서만 산출이 가능하였다. 위의 사항으로 판단한 현재의 재사용 현황은 기반 코드를 중심으로 하는 소스 코드 레벨의 재사용은 이루어지고 있으나 컴포넌트 수준의 재사용 미흡하다는 것을 파악할 수 있으며 이는 체계적인 재사용 IP(Intellectual property)의 생성 및 관리가 필요함을 보인다. 즉, 편집 설계 방법론과 재사용 저장소가 개발 보급되었음에도 실질적인 컴포넌트의 재사용이 이루어지지 않는다는 것이다. 또한 소스 코드 수준의 재사용이 활발히 이루어지고 있기 때문에 소스 코드의 구조 개선을 통한 재사용율 향상이 효과적일 것이라는 판단을 할 수 있었다. 결국 재사용 대상에 따른 적절한 재사용 방법이 제시되어야 재사용 활동이 활성화 될 수 있다는 결론을 도출할 수 있었다.

4. 적용 사례를 통한 재사용 체계의 개선

3장에서 논한 바와 같이 재사용 활동의 활성화를 위해 방법론, 저장소, 지표를 개발하고 적용하였다. 편집설계 방법론을 통해서 개발자가 수행해야 할 컴포넌트 기반의 개발을

위한 프로세스 및 세부 기법을 지침화하였으며, 이를 통해 개발된 재사용 컴포넌트를 사용, 관리할 수 있는 재사용 저장소 및 관리 시스템을 구축하였고, 재사용을 평가하기 위한 지표를 정의하여 운영하였다.

편집설계 프로세스는 기존의 소스 코드의 재사용이 재사용율은 높지만 재사용 효과가 미비하며, 비슷한 시스템을 포팅 혹은 변경하여 개발하는 경우 코딩 기간을 단축 만을 기대할 수 있고, 개발자에 따른 편차와 담당자 변경 시 대응의 어려움 등의 문제점을 해결하기 위해 컴포넌트 패키지(요구사항, 설계, 소스 코드, 시험 케이스)를 개발하고 이를 조립하는 컴포넌트 재사용 기반의 개발 프로세스 및 지침 등을 통해 재사용의 효과를 높여 요구사항에서부터 시험 케이스까지 모든 산출물을 재사용 가능하게 하고, 기 검증된 컴포넌트의 재사용을 통해 신규 소프트웨어의 품질향상을 도모하였다. 그러나 편집설계 방법론의 적용 효과는 3장의 재사용 지표 결과에서 보듯이 효과적이지 못했고 그 원인은 지금까지의 적용 경험과 개발자의 설문을 통해 다음과 같이 정리할 수 있었다.

- 소프트웨어 재사용에 대한 commitment가 부족했다는 것
- 내장 소프트웨어 특성의 반영이 부족하여 일반적인 비즈니스 소프트웨어의 컴포넌트 기반 접근 방법으로 추진하여 HW 플랫폼 및 디바이스에 대한 고민이 부족했다는 것
- 컴포넌트 자체의 개발과 재사용만을 고려하였고, 그 컴포넌트가 실행되는 실행환경으로 소프트웨어 프레임워크에 대한 접근이 없었다는 것
- 제품군 적용 측면에서의 소프트웨어 프로덕트 라인 접근방법을 포함하지 못했다는 것
- 사례를 통한 현실적인 접근 방법을 개발하지 못하고 기존의 방법론을 답습하였다는 것

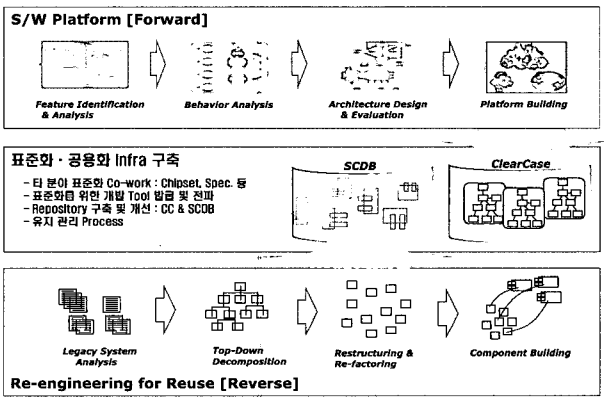
재사용 저장소(SCDB)는 초기 버전의 시스템적 문제점을 지속적으로 개선하여 1.0 버전에서 3.0까지 개발되어, 초기의 문제점이었던 형상관리 시스템 내 코드와 재사용 저장소 내의 코드 간의 일관성 문제가 해결되었다는 것이 가장 큰 성과였다. 그러나 이 역시 전사적인 차원의 소프트웨어 관리 및 제품 관리의 측면을 고려했을 때 다음과 같은 개선점을 발굴할 수 있었다.

- 제품에 적용된 전체 소프트웨어에 대한 가시화가 불가능하였다. 개발의 측면에서 형상관리 시스템이 있으나 하드웨어 B.O.M.(Bill of material : 부품표)과 같이 한 제품을 구성하는 부품의 정보나 부품 간의 관계에 대한 정보를 종합한 데이터베이스로인 소프트웨어 저장소가 부재하였다는 것
- CE 도메인의 특성 상 제품 레벨에서 하드웨어와 소프트웨어가 동일한 관리 포인트를 제공하지 못한다는 점
- 재사용의 측면에서 식별된 재사용 자산과 실제 적용되는 자산 간의 불일치가 발생한다는 것. 특히 동시에 많은 제품이 파생되어 개발되는 CE 소프트웨어 개발 환경 상 재사용 자산의 버전관리와 제품에 적용되는 소프트웨어 부품이 각각 다른 저장소에서 관리되어 이로 인한 불일치가 발생한다는 것

재사용 지표는 여러 번의 시뮬레이션 적용과 실 적용을 통해 유용함이 입증되었으나 개발 전체 조직에 적용하기 위해서는 다음과 같은 개선점이 발굴되었다.

- 제품과 관련된 전체 소프트웨어에 대한 지표가 추출되기 어렵고 또한 추출된 지표의 일관성이 보장되기 어렵다는 것
- 일부 측정 지표에 대해 툴에 의한 자동화를 수행했으나 전체 지표에 대한 자동화가 부재하였다는 것

지금까지의 재사용을 활성화하기 위해 추진한 방법론, 저장소, 지표 운영의 활동과 이의 추진을 통해 도출한 개선점을 통해서 보다 효과적인 재사용 체계를 수립하기 위해 개선된 재사용 방법론과 재사용 시스템을 구축하였다.



(그림 6) 소프트웨어 표준화/공용화 개념도

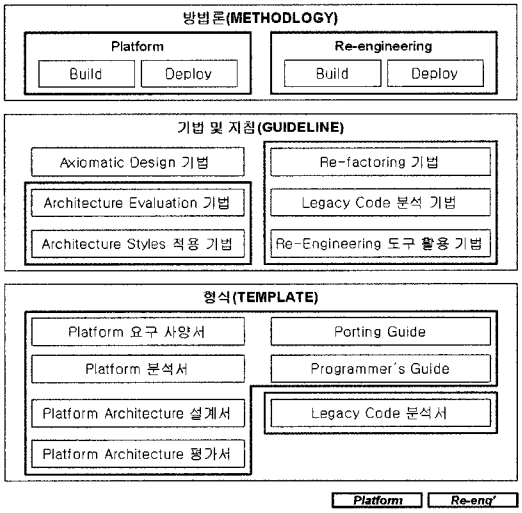
첫째, 방법론에 대한 개선은 프로덕트 라인에 대한 개념, 실행 환경인 프레임워크, 기존 코드에 대한 체계적인 재사용 방법 및 사례 반영이라는 요구사항에 의해 재사용을 표준화 및 공용화의 개념으로 정의하고 이를 구체화하는 방향으로 개선하였다. 소프트웨어 표준화 및 공용화는 소프트웨어 재사용 공학의 측면에서 파생개발에 따른 반복적이고 점증적인 개발에 대응하기 위해 소프트웨어 재사용 단위를 개발하고, 이를 이용하는 것으로 표준화 및 공용화의 재사용 단위에는 제품군에 대응하여 전체 소프트웨어의 구조 및 실행개체를 포함하는 소프트웨어 플랫폼과 기능 단위에 대응하여 특정한 기능 및 기능의 집합을 포함하는 컴포넌트로 나누어 정의한다. 이의 배경은 소프트웨어의 개발 형태가 주로 기 개발된 소프트웨어를 이용하여 추가되는 기능을 구현하거나 하드웨어의 변경에 따른 포팅 작업이 주가 되고 이는 대부분 기존 코드를 중복 사용하는 형태로 하드웨어 제품군 단위의 파생제품에 대응하는 형태를 의미하며, 이에

따라 기존 소스 코드에 대한 재사용 및 제품군 단위의 소프트웨어 개발의 효율성을 높이기 위함이다.

(그림 6)은 소프트웨어 표준화 및 공용화의 개념도로 소프트웨어 플랫폼 개발 방법은 제품군 단위의 파생제품에 대한 소프트웨어 개발을 위해, 공통기능을 추출하여 제품군별 재사용 구조를 정의하고 실행개체를 제공하는 소프트웨어 플랫폼에 대한 개념 정의와 개발 방법을 정의하며, 재공학(Re-engineering) 방법은 내장 소프트웨어 분야의 특성상 펌웨어를 주로 개발하는 소규모 소프트웨어에 대한 대응 방법으로 소프트웨어 플랫폼 개발 방법 보다는 이미 개발되어 오랜 기간 검증된 리거시 코드의 재사용을 높이기 위해, 리팩토링(Re-factoring) 등의 기법을 활용하여 코드 내부 구조 및 품질을 향상시키고 핵심 코드를 컴포넌트화하는 방법이다. 또한 방법론을 적용하기 위한 실제적인 활동으로 소프트웨어 재사용을 위한 개발 환경 구축을 포함하여 칩셋의 표준화 및 UI 표준화와 같은 관련 분야 표준화와 소프트웨어 표준화 및 공용화를 추진하기 위한 도구의 개발 및 도입을 주요 활동으로 정의하였다.

본 방법론은 (그림 7)과 같이 구성된다. 본 방법론이 차별될 수 있는 포인트는 기존의 여러 방법론 중 기업체의 현황에 가장 맞는 방법을 현황에 맞도록 제안했다는 것이며 이를 위해 전략적인 차원에서 소프트웨어 플랫폼 개발 및 재공학에 의한 기존 코드의 재사용 방법을 정의했다는 것과 방법론의 개발이 과제 적용에 대한 사례를 중심으로 개발되었다는 것이다. 이는 기존의 재사용 방법론들이 비즈니스 어플리케이션 위주로 내장 소프트웨어 영역에 대한 고려사항 및 적용사례 등이 부족하여 실제적인 적용의 한계가 있다는 것이다.

재사용 시스템에 대한 개선은 다양한 분야와 특성을 가진 제품 내장 소프트웨어를 개발하는 기업에서 소프트웨어에 대한 체계적인 관리를 수행할 수 있도록 전체 소프트웨어를 관리할 수 있는 시스템으로 그 폭을 넓히고 그 안에서 재사용 가능한 자산이 식별되어 관리될 수 있는 시스템을 구축하는 것이다. 이 시스템의 특징은 기존의 재사용 저장소가



(그림 7) 방법론 구성도

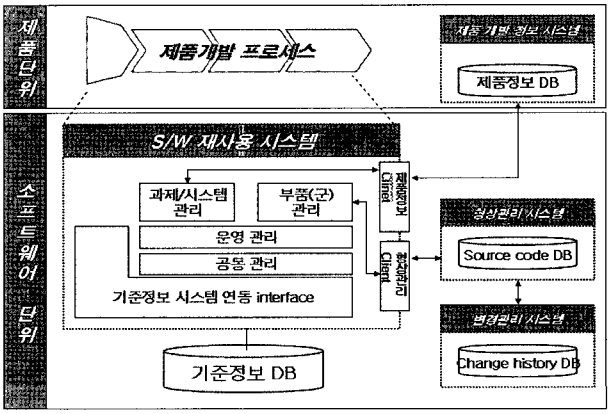
〈표 2〉 재사용 시스템의 관리 단위

분류명	정의	설명
시스템	과제(Project) 단위 과제 A 모델 A 모델 B 모델 C	하나의 과제가 많은 모델 에 대응하는 경우가 대부분이기 때문에 과제 단위로 선정함. 단, 구성 부품군 및 부품이 현격한 차이가 있을 경우 과제를 조직의 판단 하에 복수의 시스템으로 분리할 수 있음.
부품군	부품을 기능별로 분류하기 위한 S/W부품 집합	부품과 시스템 사이에 Multi-layer로 등록 가능함. 기능군으로 정의.
부품	시스템을 구성하는 최저 S/W 단위	부품의 구성요소는 형상관리에서 관리하는 File단위로 정의. 논리적인 기능단위로 여러 개의 file을 포함할 수 있음

재사용 자산(특히 컴포넌트)만을 저장, 관리하는 것에 비해 모든 코드를 부품이라는 단위로 분할하여 별도의 고유코드를 부여하여 관리하는 것이 특징이다. 일반적인 개념의 재사용 대상인 '컴포넌트'는 '부품' 중에서 재사용성이 높은(혹은 있는) 선별된 부품으로 해석할 수 있다. 본 시스템에서의 관리 대상에 대한 기준은 <표 2>같다.

재사용 시스템은 크게 세 가지의 목적을 가지고 있는데 하나는 소프트웨어 개발에 대한 가시화를 이루는 것으로 개발되는 전 소프트웨어를 식별자인 고유코드와 세부정보를 갖는 특성정보를 부여하고 구조화 및 체계화를 하여 라이브러리화 하는 것으로 과제별 소프트웨어 구조, 부품의 제품 적용 현황, 표준 및 비표준 소프트웨어 개발 현황 등을 사용자(개발자, 개발 관리자, SE 담당자, auditor, 협업 부서 등)에게 다양한 관점의 정보와 이를 활용할 수 있는 편리한 기능을 제공하여, 전사 소프트웨어 개발에 대한 표준 정보 시스템의 역할을 수행하는 것이다. 이는 하드웨어의 부품정보 시스템의 기본 개념을 차용한 것으로 궁극적으로는 제품을 구성하고 있는 하드웨어와 소프트웨어를 하나의 관점에서 관리할 수 있도록 하는 것이다.

(그림 8)과 같이 재사용 시스템의 범위는 소프트웨어 개발에 한정이다. 내장형 소프트웨어의 경우 제품개발과의 연



(그림 8) 재사용 시스템 구성도

계가 필수적이며, 이를 위해 제품 개발 정보 시스템과 연계하여 과제정보를 공유토록 하였다. 소스 코드의 부품화 관리를 위해서는 형상관리 시스템과 연동하여 부품의 논리적 정보만을 소프트웨어 정보 시스템의 데이터 베이스에서 관리한다. 즉, 물리적 코드는 형상관리 시스템에서 관리되고, 소프트웨어 정보 시스템에는 각 부품에 대한 식별 코드 및 부품 별 특성정보가 관리된다.

5. 결 론

전자업계의 소프트웨어는 대부분 모과제의 기반 코드를 재사용하여 개발하고 있는 상황에서 단일 제품 파생 계보상에서의 재사용은 잘 이루어지고 있으나, 제품 및 제품군 간 재사용이 미비한 실정이다. 본 논문은 내장형 소프트웨어에 대한 재사용성을 향상시키기 위한 기업 내 여러 사례를 소개하였고 이러한 사례를 기반으로 전사적인 재사용 체계를 정비하여 추진한 내용을 논하였다. 재사용 체계는 방법론 측면의 접근으로 내장형 소프트웨어에 대한 표준화/공용화 방법론 수립 및 적용을 시스템 측면의 접근으로 재사용 시스템을 개발하여 적용하였다.

재사용 방법론은 모과제를 기반으로 파생제품에서 재사용이 이루어지는 특성을 반영하여 제품군 차원에서 소프트웨어 재사용을 가능케하는 소프트웨어 플랫폼과 플랫폼 상에서 재사용할 수 있는 컴포넌트를 기존 코드에서 추출하는 재공학을 정의하였다.

재사용 시스템은 소프트웨어를 부품 및 부품군이라는 관리단위로 나누고 이를 관리하면서 재사용성이 높은 부품들을 재사용 지표를 통해 추출하고 이를 표준부품화 하여 관리할 수 있도록 하였다.

재사용 방법론과 재사용 시스템은 전사차원에서 적용되고 있으며, 각각 지속적인 과제적용을 통해 개선을 수행하고 있다. 이와 같은 활동을 통해 각 항목 별 개선점을 도출하고 이를 개선하는 것이 향후 중요한 연구과제가 될 것이며, 특히 지금까지 수행하지 못했던 각 분야 간 통합적인 효과분석을 추진할 예정이다.

참 고 문 헌

[1] Ir.P.N. Wouters, "Components in Embedded Software", Component technology, p.23, Jan., 1999.

[2] Carma McClure. "Software Reuse Techniques", Prentice Hall, 1997.

[3] Kangtae kim, Taesik kim, "Reinforcing the S/W development competence by the SCRM(Samsung's S/W Competence Reinforcement Model)", Proceeding of SERA 05, IEEE, 2005.

[4] Ivar Jacobson, "Software reuse : Architecture, Process and Organization for Business Success, Addison-Wesley, 1997.

[5] William B. Frakes, Kyo Kang, "Software Reuse Research : Status and Future", IEEE Transaction on Software Engineering, Vol.31, No.7, July, 2005.

[6] L. Bass, P. Clements, R. Kazman, "Software Architecture in Practice", second ed. Addison-Wesley, 2003.

[7] C. Szyperski, D. Gruntz, S. Murer, "Component Software : Beyond Object-Oriented Programming", second ed. Addison-Wesley, 2002.

[8] P. Clements, L. Northrop, "Software Product Lines : Practices and Patterns", Addison-Wesley, 2002.

[9] D.M. Weiss, C.T.R. Lai, "Software Product-Line Engineering : A Family-Based Software Development Process", Addison-Wesley, 1999.

[10] I-Ling Yen; Jayabharath Goluguri; Farokh Bastani; Latifur Khan, "A component-based approach for Embedded Software Development", Object-Oriented Real-Time Distributed Computing 2002(ISORC 2002) Proceedings, pp.402-410, April, 2002.

[11] A. Mili, R. Mili, R. Mittermeir, "Storing and retrieving software components : A refinement based system," Proc. Intl. Conf. Software Engineering, pp.91-100, May, 1994.

[12] Moormann-Zaremski, J.M. Wing, "Specification matching of software components," ACM Trans. Software Engineering and Methodology, Vol.6, No.4, pp.333-369, 1997.

[13] Luqi and J. Guo, "Toward automated retrieval for a software component respository", Proc. IEEE Conf. And Workshop on Engineering of Computer-Based Systems, March, 1999.

[14] E. Ostertag, J. Hendler, R. Prieto-Diaz, and C. Braun, "Computing Similarity in a Reuse Library System : an AI-based Approach", ACM Transactions on Software Engineering and Methodology, Vol.1, No.3, pp.205-228, July, 1992.

[15] Q. Tran and L. Chung, "NFR-Assistant : Tool support for achieving quality," IEEE Symp. Application-Specific Systems and Software Engineering and Technology, Texas, pp.284-289, March, 1999.

[16] R.A. Steigerwald, "Reusable component retrieval for realtime applications," Proc. IEEE Workshop on Real-Time Applications, pp.118-120, May, 1993.

[17] Cordeiro Pires Mascena J.C, Santana de Almeida, E.; de Lemos Meira S.R., "A comparative study on software reuse metrics and economic models from a traceability perspective", Proceeding of IRI2005(IEEE International Conference on Information Reuse and Integration), pp.72-77, August, 2005.

[18] W. Frakes, C. Terry, "Software Reuse : Metrics and Models", ACM Computing Surveys, Vol.28, pp.415-435, 1996.

[19] J. Favaro, K. Favaro, P. Favaro, "Value Based Software Reuse Investment," Annals of Software Eng. Vol.5, pp.5-52, 1998.

[20] W. Lim, Managing Software Reuse, "A Comprehensive Guide to Strategically Reengineering the Organization for Reusable Components", Prentice Hall, July, 1998.

[21] J.S. Poulin, "Measuring Software Reuse : Principles, Practices, and Economic Models", Addison-Wesley, 1997.

김 강 태

e-mail : victwin1@hanafos.com

1996년 중앙대학교 컴퓨터공학과(학사)

1998년 중앙대학교 컴퓨터공학과 소프트웨어공학(석사)

2001년 중앙대학교 컴퓨터공학과 소프트웨어공학(박사)

2001년~현재 삼성전자 책임연구원

관심분야 : 소프트웨어 프로덕트라인, 소프트웨어 아키텍쳐, 재사용 시스템