소프트웨어 개발 프로세스의 연구동향

강원대학교 권호열*

1. 서 론

소프트웨어 개발 프로세스는 소프트웨어를 생성하는 과정으로서 소프트웨어 수명주기 모델, 소프트웨어 개발 도구, 소프트웨어를 구축하는 개발자를 통합한 것으로 정의할 수 있으며[1], 소프트웨어공학 분야가 시작된 이래 제한된 시간과 비용으로 최선의 품질을 갖는 소프트웨어를 개발하기 위하여 꾸준히 연구되어 왔다.

먼저 소프트웨어 수명주기 모델은 요구분석, 설계, 구현, 테스트, 유지보수 과정을 순차적으로 거치는 전통적인 폭포수 모델 외에 프로토타이핑 모델, 반복적인 나선형 모델, 객체지향 모델 등으로 연구가 진행되어 왔으며[2-5] 최근에는 개발 반복주기를 극한적으로 단축시킨 XP(eXtreme Programming)[6] 이 소개되어 많은 관심을 끌고 있다. 또한, 소프트웨어의 개발 기간을 단축하고 품질을 안정시키기 위한 재사용 기법도 비교적 단순한 코드 재사용[7]으로 시작하여 객체 지향 시스템의 객체 재사용과 프레임워크[8-9], 컴포넌트 재사용[10,11], 설계 패턴과 아키텍처의 재사용[11-12], 프로덕트 라인 기법[13]으로 발전하고 있다. 이와 아울러 프로세스 개선과 능력 측정에서 조직차원의 관리기법인 ISO 9000[14], 소프트웨어 프로세스의 능력 성숙도 모델(CMM)[15]과 소프트웨어 프로세스 개선 및 능력 결정 모델(SPICE)[16] 등이 하나의 주류를 형성하며 연구되고 있으며, 최근에는 개인 및 팀 단위의 소프트웨어 개발 프로세스를 개선하기 위한 개인 소프트웨어 프로세스(PSP)[17]과 팀 소프트웨어 프로세스(TSP) 기법[18]이 소개되어 주목받고 있다.

본 고에서는 소프트웨어 개발 프로세스의 주요 기법으로서 소프트웨어 재사용 기법인 컴포넌트 기반 개발기법과 아키텍처 및 프로덕트 라인 개발기법, 소프트웨어 수명주기 모델로서 극한 프로그래밍(XP), 프로세스 관리 기법으로서 PSP/TSP 와 CMM 등을 중심으로 연구 동향을 간략하게 소개한다.

2. 소프트웨어 재사용 프로세스

2.1 컴포넌트기반 개발 프로세스

그림 1은 소프트웨어 시스템을 개발하기 위하여 필요한 소프트웨어 프로세스와 컴포넌트의 관계를 보여준다. 시스템을 구성하는 사용자 인터페이스, 사용자 응용 프로그램, AI 및 에이전트, OS/DBMS 및 미들웨어, 네트워크 등의 컴포넌트와 시스템의 정의 및 개발, 컴포넌트 선택, 통합 및 확인 프로세스를 이끌어 갈 소프트웨어 프로세스가 아키텍처, 합성 프레임워크 및 소프트웨어공학적인 원칙아래 정의, 개발, 시험, 검증 및 사용평가 도구의 지원을 받아 결합하여 하나의 완성된 시스템을 구축하고 있다.

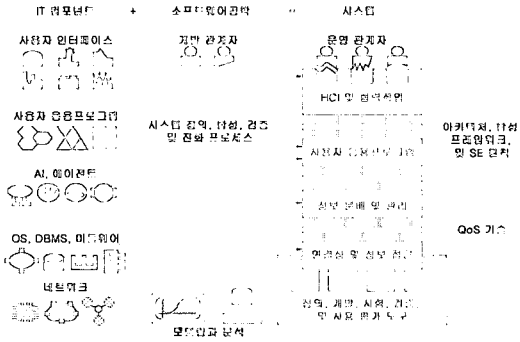

그림 1 컴포넌트와 소프트웨어 프로세스의 결합에 의한 시스템 개발[19]

* 종신회원

특정한 소프트웨어를 개발할 때 기존의 다른 소프트웨어로부터 얻어진 컴포넌트를 다시 활용하는 소프트웨어 재사용 (Software Reuse) 기법은 우수한 품질의 프로덕트를 제한된 예산과 기한 내에 얻을 수 있는 효과적인 방법으로 기대되어 왔다. 70년대부터 시작된 소프트웨어 재사용의 연구는 초기의 주로 COBOL, FORTRAN 으로 작성된 코드 모듈 또는 라이브러리의 재사용으로부터 80년대 후반의 객체지향 기법에 의한 객체 및 컴포넌트의 재사용, 90년대의 설계패턴와 아키텍처의 재사용에 이르기까지 점차 소프트웨어 개발 프로세스의 상위 수준을 향해서 확대되어 왔다.

컴포넌트에 기반한 개발 방법은 객체지향 시스템과 함께 시작되었으며, 여기서 컴포넌트는 "잘 정의된 아키텍처 내의 명확한 기능을 갖고 있으며 독립적이고, 대치가능한 시스템의 일부분"[20] 또는 "독립된 생산, 획득, 및 설치하며 상호작용하는 기능적 시스템을 형성하는 단위"[21]로 정의된다. 컴포넌트에 기반한 개발 방법은 UML 과 EJB 등 객체지향 기술에 크게 의존하고 있으며, 최근 Visual Basic, C++, JavaBeans 등 컴포넌트 기반 응용 소프트웨어의 개발을 지원하는 프로그래밍 언어와 개발환경의 출현과 함께 발전을 거듭하여 OMG의 Corba, Sun의 JavaBeans 및 EJB, Microsoft의 COM (Component Object Model) 및 Distributed COM를 중심으로 표준화 되고 있다.

컴포넌트 기반 개발 방법이 전통적 소프트웨어 개발과 구별되는 점은 다음과 같다.

- 컴포넌트 기반 프로세스는 인터페이스 중심 프로세스로서 컴포넌트를 결합시키는 컴포넌트 인터페이스가 주요 역할을 수행한다.
- 개발 프로세스는 일종의 합성 장치로서 개발자는 컴포넌트로부터 응용 시스템을 조립한다.
- 컴포넌트로부터 개발된 응용 시스템은 개별 컴포넌트의 관심과 기능성의 분리에 크게 의존적이다.
- COTS (Commercial Off-the-Shelf) 컴포넌트를 사용하여 구성된 응용소프트웨어는 판매자와 그의 업그레이드 일정에 종속적이다.

현재 논의되고 있는 컴포넌트 기반 개발 방법의 주요 연구 현안은 그림 2와 같다. 이것은 프로세스, 프로덕트 및 사람에 대한 현안을 각각 컴포넌트 제공자, 컴포넌트 통합자, 및 컴포넌트 기반 시스템 사용자의 관점에서 정리한 것으로서 사람에 대한 현안은 소프트웨어 개발 기술 현안과 비 기술적인 업무 현안으로 구분하였다.

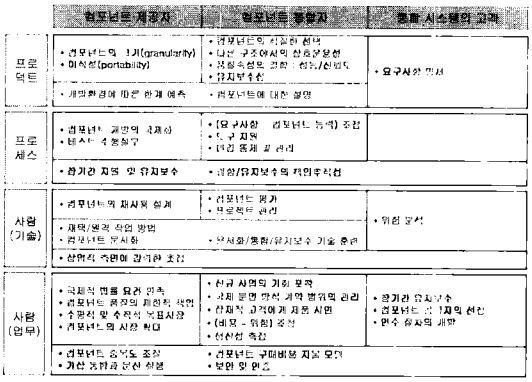

그림 2 컴포넌트 기반 시스템의 주요 연구 현안[21]

2.2 COTS 기반 개발 프로세스

컴포넌트 기반 개발 방법의 특별한 형태로서 상점에 진열되어 있는 선반위의 판매용 기성품 컴포넌트 (COTS : Commercial Off-the-Shelf)를 이용하는 COTS 기반 개발 방법은 기성품이 이미 완성되어 검증된 것이라는 점에서 프로그래밍과 테스트에 사용되는 노력이 크게 절감될 것으로 기대되고 있다. COTS 프로덕트는 다음과 같이 정의할 수 있다.

- 일반 공중용으로 판매, 대여 또는 허가되며, 수익을 얻으려는 판매자에 의해 제공된다.
- 지식 재산권을 보유하고 있는 판매자에 의해 지원되고 계속 진화된다.
- 다수의 동일한 복사본이 입수 가능하며, 원시 코드의 수정없이 사용된다.

위와 같은 COTS 프로덕트의 특성에 따라 COTS 기반 시스템을 개발하는 프로세스의 특성이 달라진다. COTS 기반 개발 프로세스는 요구분석, 아키텍처 설계, 구현 등의 과정이 순차적으로 진행되는 전통적인 개발방법과 달리 시장에서 구입가능한 부품들 가운데서 요구분석과 아키텍처 설계의 조건을 만족하는 것들을 선택하여 구매 후 일괄 조립하는 방식으로서 일종의 합성(composition) 활동이다. 따라서, 현실적으로 형성되어 있는 COTS 프로덕트 시장이 COTS 기반 시스템의 개발의 가능성 여부를 좌우하며, COTS 기반 시스템의 개발은 그림 3에서 보는 바와 같이 COTS 프로덕트 시장 상황, 시스템 아키텍

처 및 설계, 시스템 요구사항 들이 동시에 만족되는 조건에서 결정된다.

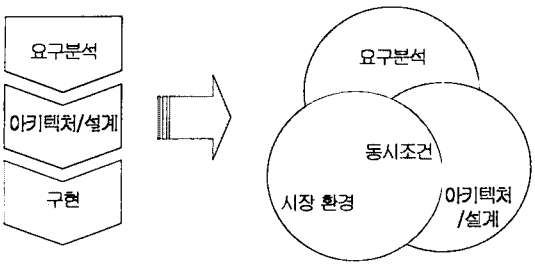

그림 3 전통적 개발 방법(a)과 COTS 기반 개발 방법(b)

COTS 기반 개발 프로세스는 아직 확립된 프로세스 모델을 갖고 있지 못하며, 이에 대한 연구가 필요하다.

Brownsword 등[22]의 제안에 따르면, COTS 기반 프로세스는 공학(engineering), 업무(business), 프로젝트(project-wide), 계약(contract)이라는 4개의 주요 활동 영역(activity area)으로 구분하며 COTS 기반 프로세스의 최상위 구조를 결정하는 4개의 활동영역 중에서 공학, 업무, 계약 범주는 각각 단일 영역에 속한 활동을, 프로젝트 범주는 여러 영역에 걸쳐 있는 활동을 대표한다. 공학 활동 영역은 시스템의 기술적인 개념화, 구현, 및 유지보수에 관련되며, 그림 3의 COTS 개발 방법을 구현한 것으로서 요구사항, 시장, 아키텍처와 설계 활동은 동시에 서로 협력적으로 수행되어야 한다. 업무 활동 영역은 업무 사례와 비용 산정을 개발하고, 라이센스 협상 및 공급자 관리를 지원한다. COTS 프로덕트와 기술적 결정은 공학적인 측면과 사업적 측면을 함께 고려해야 한다. 업무 활동영역의 많은 활동이 공학 활동 영역과 상호 정보 교환을 요구한다. 프로젝트 활동 영역은 COTS 기반 시스템의 개발 및 유지보수와 함께 공학적 및 업무적 활동영역을 넓히고 통합한다. COTS 기반 프로세스의 활동 영역 가운데 일부를 그림 4에 보였다.

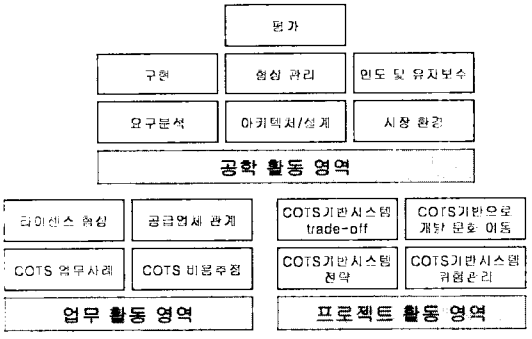

그림 4 COTS 기반 소프트웨어 프로세스의 활동 영역[22]

2.3 아키텍처 기반 개발 프로세스

소프트웨어 아키텍처는 "프로그램/시스템의 구성요소 및 이들간의 상호관계의 구조, 그리고 이들의 설계와 시간에 따른 진화를 지배하는 원리와 지침"으로 정의된다[23]. 소프트웨어 아키텍처의 개발은 대규모 소프트웨어 시스템의 개발에서 핵심적인 단계로서 서로 다른 기능성을 갖는 여러개 시스템이 동일한 기초 아키텍처로부터 생성되는 소프트웨어 프로덕트라인의 개발에 기본이 된다. 아키텍처의 개발 과정은 아키텍처 요구사항의 도출, 아키텍처 설계, 아키텍처 문서화, 아키텍처 분석, 아키텍처 구현, 아키텍처 유지보수 등 6개 단계를 거치며 이 가운데 설계, 문서화, 분석 단계는 반복적으로 수행된다. 이들 각 단계는 정보를 수집하는 수단을 포함하는 입력물, 건설적 활동, 유효화 활동, 출력물을 포함한다.

일단 사용 가능한 아키텍처가 얻어진 후에, 전통적인 소프트웨어 개발 프로세스의 수명주기로 돌아가서 소프트웨어의 구현이 시작되며 테스트 및 인수 단계를 거쳐 최종적으로 유지보수 단계가 진행된다. 그림 5는 아키텍처 기반 프로세스를 나타낸 것이며 그림 6은 소프트웨어 아키텍처에 대한 프로그래머, 시스템 엔지니어, 시스템 통합자, 최종사용자의 입장에서 본 4가지 관점을 나타낸 것이다.

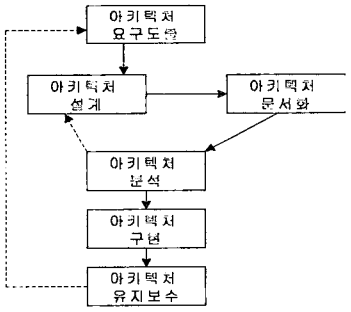

그림 5 아키텍처 기반 개발 프로세스[23]

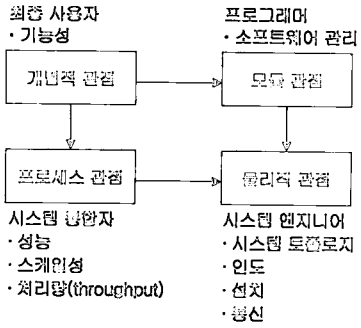

그림 6 아키텍처의 관점[24]

오늘날 체계적인 소프트웨어 재사용의 주도적인 형태는 소프트웨어 프로덕트 라인[13, 14]으로 대표된다. 소프트웨어 프로덕트 라인의 핵심은 다양한 최종 프로덕트를 지원하는 하나의 공통 아키텍처이다. 프로덕트 라인에서 재사용은 응용시스템이 속한 영역(domain)에 대해 접근하여 시장 측면에서 구분된 영역에 의해 생성된다. 시스템 고유의 소프트웨어 아키텍처에 관련된 모든 연구 현안들은 프로덕트라인 아키텍처에도 동일하게 적용되며 다음과 같다.

- 아키텍처 기술 언어
- 소프트웨어 아키텍처 평가 기법
- 소프트웨어 아키텍처를 진화시키면서 무결성을 유지하는 방법
- 하드웨어와 시스템 수준 아키텍처를 소프트웨어 아키텍처와 통합하는 방법
- 컴포넌트와 시스템의 다수 버전을 효율적으로 관리하는 형상관리 방법

아키텍처와 컴포넌트의 일부 수정 및 교체를 통한 효과적인 시스템 업그레이드 방법

3. 극한 프로그래밍(XP)

소프트웨어 프로세스 모델에 있어서 전통적인 폭포수 모델은 사용자가 개발자에게 한 번에 모든 요구사항을 정확하게 전달하는 것을 가정하고 있으나, 현실적으로 사용자들은 자신들이 심지어 무엇을 원하는 지 정확히 모를 뿐 만 아니라 요구사항을 한 번에 모두 이야기하지도 않고 자주 마음을 바꾸기도 한다. 소프트웨어 개발의 수명주기를 짧게 반복하는 반복형 모델은 요구사항의 변경에 대한 소프트웨어 프로세스의 적응성을 개선하는 효과를 얻을 수 있으며, 극한 프로그래밍(XP : eXtreme Programming)은 그림 7에서 보는 바와 같이 반복형 모델의 개발 주기를 극단적으로 짧게 함으로써, 프로그래머가 설계, 구현, 시험 활동을 전체 소프트웨어 개발 기간에 걸쳐 조금씩 자주 실시하도록 하고 소프트웨어 변경의 비용을 최소화하는 기법이다.

XP의 특징적인 주요 수행활동(Practice)을 살펴보면, 개발자의 추정자료를 이용하여 고객이 소프트웨어 공개(release)의 범위와 시기를 정하는 "게임 계획", 공개를 소규모로 자주하는 "소규모 공개", 고객과 개발자 간의 은유에 의해 시스템의 형상을 정하는 "메타포(metaphor)", 최소한의 클래스와 메소드만 사용하는 "단순 설계", 개발자가 매 순간 단위시험을 실시하고 고객은 반복 단위 기능시험을 실시하는 "시험", 모든 시험을 유지하면서 기존 설계를 진화시키는 "재구성(Refactoring)", 2인이 1대의 컴퓨터를 사용하여 코딩하는 "짝(pair) 프로그래밍", 코드가 새로 개발되면 즉시 통합하는 "연속 통합", 모든 개발자가 모든 코드를 책임지는 "집합적 소유권", 고객이 개발팀에 상주하는 "현장 고객", 최고의 업무 효율을 위한 작업시간 제한의 "주당 40시간", 작업장 내 트인 중앙부분에서 통합 작업을 하는 "개방된 작업공간" 등이 있다. 표 1은 전통적 프로세스와 XP 프로세스를 비교한 것이다.

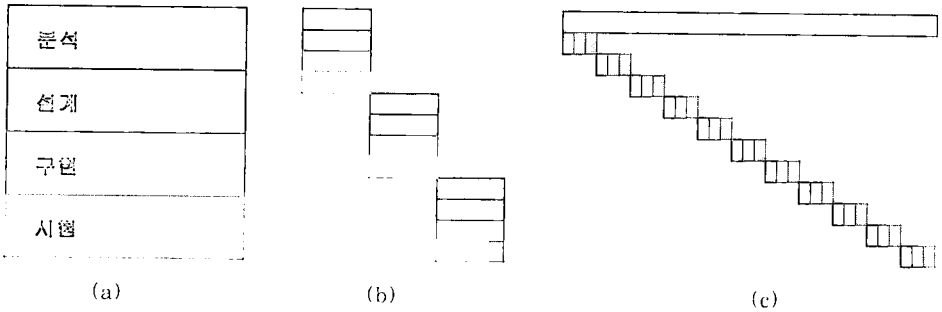

그림 7 개발 프로세스 모델 (a) 폭포수 모델 (b) 반복형 모델 (c) XP 모델

표 1 XP의 극한 프로세스[25]

전통적 과정	XP 극한 프로세스	XP 구현 실무
코드검토	코드검토를 항상 수행	짝(pair) 프로그래밍
시험	(고객에 의한) 시험을 항상 수행	단위 시험, 기능 시험
설계	설계를 모든 사람의 일상업무화	설계 재구성(Refactoring)
단순성	시스템의 현재 기능을 지원하는 가장 단순한 설계를 이용한 작업	작동 가능한 가장 단순한 것
아키텍처	아키텍처 재정의 작업에 모든 사람이 항상 참여	메타포어
통합시험	하루에도 여러번 통합 및 시험 수행	연속 통합
반복주기 단축	작업 반복주기를 주/월/년 단위에서 초/분/시 단위로 극히 단축	게임 계획

4. 소프트웨어 프로세스 개선

4.1 PSP와 TSP

소프트웨어 개발 프로젝트는 품질 문제, 비용과다, 및 일정 지연으로 항상 문젯거 발생하며, 이러한 문제는 부분적으로 개인적 수준 또는 개발팀의 역량에 기인한다. 개인 소프트웨어 프로세스(PSP)와 팀 소프트웨어 프로세스(TSP)는 각각 개발자 개인 또는 개발팀의 소프트웨어 프로세스를 평가하고 개선하기 위한 모델이다.

PSP를 사용할 때 개발자는 자신의 작업을 계획하고, 진행중인 작업을 측정하며, 작업이 끝났을 때 그들의 성능을 개선시키기 위하여 측정치를 분석함으로써 궁극적으로 개인과 개인이 속해있는 조직의 생산성을 향상시키도록 한다. PSP는 개발자가 훈련된 개인 소프트웨어 프로세스를 구축하도록 돕는 잘 정의된 프레임워크를 제공한다. 그림 8은 PSP 에서 정의된 프로세스 모델로서 프로젝트 수명주기 활동이 계획, 설계, 설계검토, 코드, 코드검토, 컴파일, 테스트, 및 사후조치 등 시간의 흐름에 따른 단계로 나뉘어 있다. 가늘은 화살표는 개인이 결함을 검출하는 단계에서 출발하여 개인이 결함을 주입한 단계로 추적하는 과정을 가리킨다. 굵은 화살표는 산출물의 흐름을 가리킨다. 사후조치에서 계획으로 되돌아가는 점선은 PSP 과정에서 얻어진 경험을 후속 프로젝트로 전달하는 것을 나타낸다.

PSP에서 사용되는 개인 소프트웨어 프로세스의 성능을 측정하기 위한 속성의 예는 표 2와 같다. 만일 최근 개발 단계에 포함된 소프트웨어 결함을 의미하는 Abtk 값이 낮다면 상대적으로 쉽게 결함을 제거할 수 있으므로 높은 생산성과 낮은 개발 비용이라는 결론을 얻을 수 있다.

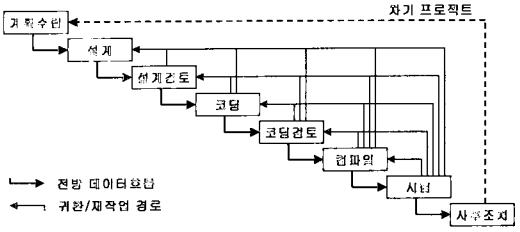

그림 8 개인 소프트웨어 프로세스(PSP) 모델[12]

표 2 프로세스 속성[26]

속성	정의
Dds	개발된 코드의 1000줄 당 제거된 결함의 수
Drr	시간당 제거된 결함의 수
생산성	시간당 개발된 코드의 줄 수
A/FR	컴파일/테스트에 시간 대비 설계검토/코드검토 시간의 백분율
수율(Yield)	최초 컴파일 이전에 제거된 결함의 백분율
Abtk	결함수정을 위해 후진한 프로세스 단계의 평균 수
DT/TDT	전체 개발 시간 대비 설계 단계에 사용된 시간
TT/TDT	전체 개발 시간 대비 테스트 단계에 사용된 시간
NTD/ND	전체 결함의 수 대비 테스트에서 검출된 결함의 백분율

4.2 CMM와 SPICE 모델

대표적인 소프트웨어 프로세스 심사규격은 모든 산업에 포괄적으로 적용되는 ISO 9000 품질 시스템[14], 소프트웨어 분야에 전문화된 미국의 CMM

(Software-Capability Maturity Model) 규격[15] 및 CMM의 기존 개념을 국제 표준으로 확장한 ISO 15504 규격 (또는 SPICE 모델)[16]이 있다. 특히 소프트웨어적인 관점에 제한적인 CMM은 정보시스템 공학적인 관점으로 확장된 CMMI[27]로 변환되고 있으며 국내에서도 중소기업청의 주도로 정보화경영체제(IMS) 규격[28]이 제정되어 기업의 정보시스템의 심사를 시행하고 있다.

미국 CMU/SEI에서 개발된 소프트웨어 프로세스의 능력 성숙도 모형인 SW-CMM은 1991년에 SW-CMM V1.0이 발표된 후 거듭 발전해 오다가 2000년에 미국 국방성(DoD)과 방위산업협회(NDIA)의 지원아래 SW-SMM, EIA/IS 731 SECM(System Engineering Capability Model), IPD-CMM(Integrated Product Development-CMM) 등 3개 규격이 통합되어 소프트웨어, 시스템, 프로덕트를 모두 지원하는 CMMI (CMM-Integration)로 확장 발전하게 되었다.

CMMI는 소프트웨어공학(SW), 시스템공학(SE), 통합프로젝트관리(IPPD), 도입(A) 등 분야에 각각 다른 규격을 적용할 수 있도록 설계되어 있다. 예를 들어 모든 분야를 다 포함하는 경우는 CMMI-SW/SE/IPPD/A[29]를 적용하도록 되어 있다. CMMI의 프로세스 영역은 프로세스 관리, 프로젝트 관리, 설계/개발, 지원, 도입 등 5개 프로세스 영역으로 나뉘어져 있으며, 각 프로세스 영역마다 여러 개의 프로세스 및 이에 부속되는 목적과 실무절차 들이 존재한다. CMMI 는 기존에 사용하던 심사모형이 EIA /IS 731 인지, 또는 SW-CMM 인지 여부에 따라 그림 9(a)의 연속적 표현(Continuous Repres.) 모형, 또는 그림 9(b)의 단계적 표현(Staged Repres.) 모형을 사용할 수 있다.

한편, 소프트웨어 심사를 위한 대표적인 국제표준인 ISO 15504는 공급, 개발, 지원, 관리, 조직 등의 소프트웨어 프로세스에 대한 계획, 관리, 감시, 통제, 개선을 위한 능력심사와 프로세스 개선을 목적으로 하며 일명 SPICE(Software Process Improvement and Capability dEtermination)라고 한다. 소프트웨어공학 표준화 그룹인 ISO/IEC JTC1/SC7/ WG10에서 1992년 1월부터 표준화 작업을 시작한 SPICE는 국제표준 승인전 최종단계인 "Type 2" 규격이 1998년 발표되었으며, 세계 각국에서 산업체 적용 실험을 통하여 2002 년에는 정식 표준으로 등록 될 예정이다.

SPICE 심사는 프로세스 개선과 능력 결정이라는

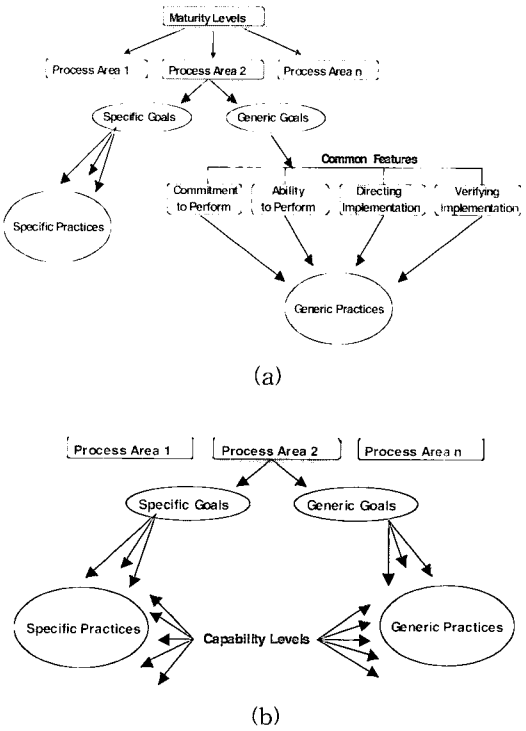

그림 9 CMMI 심사모형 (a) 연속적 표현 (b) 단계적 표현[27]

두 가지 목적 아래 실시된다. 프로세스 개선은 경영관점에서 조직의 현행 실무절차가 갖는 강점과 약점, 프로세스 고유의 위험요소를 식별함으로써 조직의 목표를 달성하기 위한 효율성과 품질, 납기, 비용 측면의 문제점을 찾아내도록 한다. 능력 결정은 진행중인 프로젝트에 내재된 위험을 식별하기 위해서 선택된 프로세스의 특성을 목표치에 비교하여 평가한다.

SPICE의 심사모형은 그림 10에서 보는 바와 같이 프로세스 차원과 능력 차원이라는 2개의 차원으로

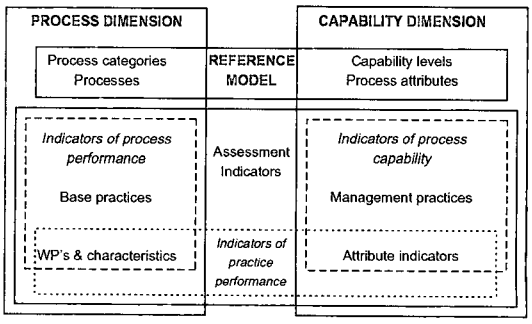

그림 10 SPICE의 프로세스심사 모형[16]

구성되며 각각 소프트웨어 개발 수명주기상의 프로세스가 적절히 구성되었는 지, 또는 피심사 프로세스가 Level 0 에서 Level 5에 이르는 6가지 능력수준 가운데 어느 것에 해당하는 지 측정하고 프로세스의 개선점을 찾아내는 데 사용된다.

4.3 정보화 경영시스템(IMS)

국내에서 정부주도로 제정된 정보시스템 심사규격인 IMS(Information Management System)시스템은 기업의 경쟁력이 정보화를 통한 생산성과 효율성에 크게 의존하고 있다는 인식에서 출발한 것으로 조직의 경영목표 달성을 위해 소프트웨어 개발과 구매를 포함하는 기업 정보화를 추진함으로써 경영혁신을 이루도록 하는 것이 특징이다.

1999년부터 연구가 시작된 IMS 규격은 ISO 9000/ISO 14000 및 SPICE 등 정보시스템 심사관련 규격들을 참고하여 초안이 작성되었으며, 2000년 30개 기업의 시범사업과 초안 수정을 거쳐 2000년 12월 IMS V1.0이 발표되었다. 이어 2001년 5월에 IMS 관련 법률인 중소기업기술혁신촉진법이 공포되어 법적인 근거를 갖는 정보화 시스템 인증 제도로 정착하게 되었다.

IMS 시스템은 그림 11에서 보는 바와 같이 정보화 경영방침, 계획, 도입 및 확장, 운영, 점검 및 시정조치 등 5가지 프로세스 영역으로 구성된 정보시스템 수명주기에 따라 심사를 진행하며, 각 프로세스 영역에 존재하는 세부 프로세스는 IMS 심사지침에 따라 작성된 체크리스트를 이용하여 2,300 점을 만점으로 평가하며, 각 프로세스 영역에 따른 조직의 정보화경영체제에 대한 성숙도의 평가결과는 백분율로 환산했을 때 50% 이상을 수준1 (L1), 60% 이상을 수준2 (L2)로 하는 등, 90%를 넘는 경우 최고 등급은 수준5 (L5)로 판정한다.

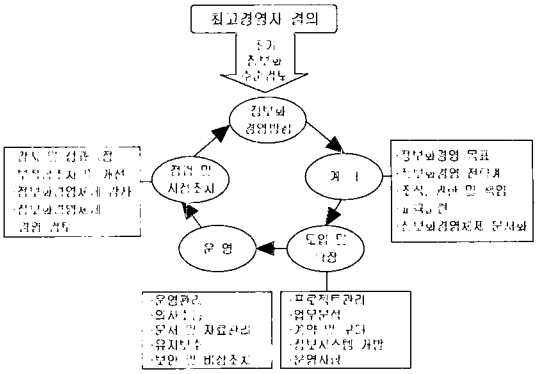

그림 11 IMS의 심사 모형[28]

표 3은 앞에서 살펴본 심사규격의 능력수준판정에 대한 비교표이다. 앞에서 살펴 본대로 각 심사규격의 프로세스 집합의 구성과 정의는 다소 다르지만 전체적인 등급 판정기준은 거의 같은 형태를 유지하고 있다.

5. 결 론

앞에서 소프트웨어 개발 프로세스의 최근 연구 동향은 소프트웨어 재사용 프로세스와 극한 프로그래밍, 그리고 프로세스 개선을 위한 심사 시스템의 순서로 간략하게 살펴 보았다. 본 고에서 다룬 주제와 관련하여 향후 주목해야 할 소프트웨어 프로세스의 발전 방향은 다음과 같다.

첫째, 현재 아키텍처 및 프로덕트 라인 형태의 재사용이 소프트웨어 수명주기 위의 보다 높은 추상화 수준인 요구사항의 재사용으로 진행될 수 있을 것이며, 이것은 컴포넌트 및 아키텍처의 재사용이 충분히 성숙한 단계 위에서 본격적으로 논의가 가능하게 될 것이다.

둘째로 수명주기의 반복기간이 극히 단축된 형식의 XP 프로그래밍은 설계, 구현, 테스트가 조기에 동

표 3 능력수준의 판정

능력 수준	CMM		SPICE	Korean IMS
	연속적 표현	단계적 표현		
Level 5	Optimizing	Optimizing	Optimizing	수준5
Level 4	Quant'ly Managed	Quant'ly Managed	Predictable	수준4
Level 3	Defined	Defined	Established	수준3
Level 2	Managed	Managed	Managed	수준2
Level 1	Performed	Initial	Performed	수준1
Level 0	Incomplete	N/A	Incomplete	불합격

시에 시작되어 함께 진행된다는 점에서 기존의 전통적인 폭포수 모델이 갖는 충분히 공들인 설계가 전반적인 소프트웨어 개발 프로세스의 효율을 높인다는 개념과 다소 다른 형태를 갖고 있다. 이러한 다른 특성을 갖는 개발 프로세스 모델에 가장 적합한 개발환경 또는 프로젝트 특성의 관계에 대한 연구도 필요하다.

끝으로 프로세스 개선 및 심사에 대해서는 현재 국내에서 소프트웨어 프로세스에 대한 SPICE와 CMM에 대한 연구가 활발하게 진행되고 있으며, 점차 대형화되어 가는 시스템 안에 소프트웨어가 기본적으로 내장(Embedded)되는 경우가 많아짐에 따라 하드웨어까지 통합하는 시스템 관점의 프로세스 심사에 대한 CMMI 및 IMS 등에 대한 연구가 요구된다.

참고문헌

[1] S.R. Schach, Object-Oriented and Classical Software Engineering, 5th Ed., McGraw Hill, 2002.

[2] B. Boehm, Software Engineering Economics, Prentice Hall, 1981.

[3] B. Boehm, "A Spiral Model of Software DEvelopment and Enhancement," IEEE Computer, May 1988, pp. 61-72.

[4] K.E. Lantz, The Prototyping Methodology, Prentice Hall, 1985.

[5] I. Jacobson, M. Christerson, P. Jonsson, and G. Overgaard, Object Oriented Software Engineering: A Use Case Driven Approach, ACM Press, 1992.

[6] K. Beck, "Embracing Change with Extreme Programming," IEEE Computer, Vol. 32, No. 10, Oct. 1999, pp. 72-77.

[7] R.G. Lanergan and C.A. Grasso, "Software Engineering with Reusable Design and Code," IEEE Tr. on Software Engineering, TSE 10, Sep. 1984, pp. 498-501.

[8] T. Isakowitz and R.J. Kauffman, "Supporting Search for Reusable Software Objects," IEEE Tr. on Software Engineering, TSE 22, June 1996, pp. 407-423.

[9] S. Sparks, K. Benner, and C. Faris, "Managing Object Oriented Framework Reuse," IEEE Computer 29, Sep. 1996, pp. 52-61.

[10] R. Mili, A. Mili, R.T. Mittermeir, "Storing and Retrieving Software Components: A Refinement Based System," IEEE Tr. on Software Engineering, TSE 23, July 1997, pp. 445-460.

[11] D.C. Schmidt, "Using Design Patterns to Develop Reuseable Object Oriented Communications Software," Comm. of ACM 38, Oct. 1995, pp. 65-74.

[12] M. Shaw and D. Garlan, Software Architecture: Perspective on an Emerging Discipline, Prentice Hall, 1996.

[13] G. Chhastek, P. Donohoe, K.C. Kang, S. Thiel, Product Line Analysis : A Practical Introduction, CMU/SEI-2001-TR-001, June 2001.

[14] ISO 9001:2000, Quality Management Systems - Fundamentals and Vocabulary.

[15] CMU/SEI, Capability Maturity Model for Software V1.1, Draft C (SW-CMM), 1993.

[16] ISO/IEC 15504-1998, Software Process Assessment, 1998.

[17] W.S. Humphrey, Introduction to the Personal Software Process(sm), SEI, 2000.

[18] Team Software Process Executive Seminar, SEEK/ISRI, June 19, 200.

[19] B. Boehm and V.R. Basili, Gaining Intelligent Control of Software Development, IEEE Computer, Vol. 33, No. 5, May 2000, pp. 27-33.

[20] A.W. Brown and K.C. Wallnau, The Current State of CBSE, IEEE Software, Vol 16, No. 5, pp. 37-46, Sept./Oct. 1998.

[21] P. Brereton and D. Budgen, Component-Based Systems: A Classification of Issues, IEEE Computer, Vol. 33, No. 11, pp. 54-62, Nov. 2000.

[22] L. Brownsword, Tricia Oberndorf, and C.A. Sledge, "DEveloping New Processes for COTS Based Systems," IEEE Software, July/Aug. 2000, pp. 48-55.

[23] L. Bass and R. Kazman, Architecture Based Deveopment, CMU/SEI 99-TR-007, April 1999.

[24] P.C. Clements and L.M. Northrop, Software Architecture: An Executive Overview, Tech. Report CMU/SEI 96 TR 003, Feb. 1996.

[25] M.C. Paulk, Extreme Programming from a CMM Perspective, IEEE Software, Vol. 18, No. 6, pp. 19-26, Nov./Dec. 2000.
[26] X. Zhong, N.H. Madhavji, and K.E. Eman, Critical Factors Affecting Personal Software Processes, IEEE Software, Vol. 18, No. 6, pp. 76-83, Nov./Dec. 2000.
[27] CMU/SEI, Capability Maturity Model Integration (CMMI), 2000.
[28] 정보화경영체제(IMS) 규격 V1.0, 중소기업청, 2000.
[29] CMU/SEI, CMMI for systems Engineering/Software Engineering/Integrated Pro duct and Process Development/Acquisition (CMMI-SW/SE/IPPD/A) V1.02d Draft, 2001.

권 호 열

1982 서울대학교 전자공학과 공학사
1982 한국과학기술원 전기전자공학과 공학석사
1991 한국과학기술원 전기전자공학과 공학박사
1991 한국통신 연구개발단 선임연구원
1995~1996 미국 Stanford 대학교 방문학자
2000 미국 CMU 대학교 ASE과정 연수
1991~현재 강원대학교 전기전자정보통신공학부 부교수
관심분야:Software Process, PLSE, CMM/SPICE/IMS
Email:hykwon@kangwon.ac.kr