해 설

소프트웨어 형상 관리와 보수 관리

박 정 호† 양 해 술††

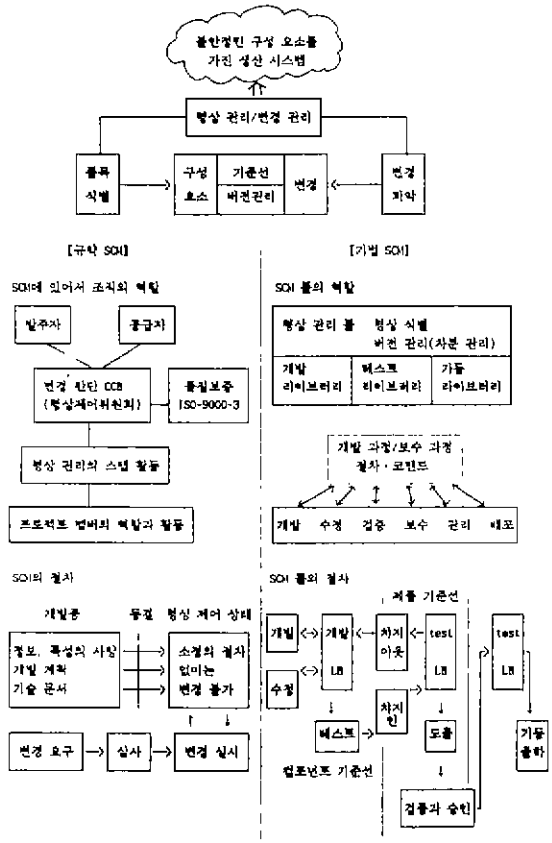
❖ 목 차 ❖

1. 서 론

소프트웨어 개발과 보수의 생산 관리 기술로서 소프트웨어 형상 관리(Software Configuration Management : 이후 SCM이라고 한다)가 오늘날 많은 주목을 받고 있다[1, 2]. SCM이란 불안정한 구성 요소를 가진 시스템 즉, 변경이 일어나는 생산 형태에 있어서의 관리 기술을 말한다.

(그림 1)에 형상 관리의 전체 개요를 나타냈는데, SCM에는 세가지 흐름이 있다. 하나는 미국에서의 대규모 시스템 개발의 구입자에 의해 개발된 하향식 관리 절차로서, 발주자와 공급자간의 계약을 토대로 해서 발주자와 공급자가 변경을 합리적으로 다룰 수 있도록 한 규약이다. 하향식 관리 절차는 (그림 1)의 왼쪽에 해당하는 것으로서, 이것을 규약 SCM이라 부르기로 한다. 이미 미국에서는 ANSI/IEEE의 표준으로서 제정되어 있으며, 현재 국제 표준으로서 ISO/IEC JTC 1 SC7에서 심의중이다[3, 4, 5].

SCM은 품질보증의 국제표준인 ISO-9000시리즈 및 그 인증제도 중에서도 중요한 역할을 한다.



(그림 1) 형상 관리의 전체 개요

† 종신회원 : 선문대학교 전자계산학과 교수
†† 종신회원 : 한국정보처리학회 총무이사

또하나의 SCM은 프로그램 개발과 변경을 다루는 상향식 기법으로서, 이것을 기법 SCM이라고 한다. 기법 SCM은 (그림 1)의 오른쪽에 해당하며, 라이브러리 관리에서 시작되었다. 소프트웨어란 많은 구성 요소(소스 모듈, 데이타 정의, 마크로, 컴파일 옵션 등)로 이루어지며, 소프트웨어를 생산하기 위해서는 많은 인원이 투입된다. 이들 소프트웨어의 구성 요소는 계속해서 변경되는데, 변경될 때마다 시스템을 재조립하게 된다. 기법 SCM이란 이와 같은 혼란스러운 생산 활동을 지원하는 툴과 관리용 틀이다[7, 8].

규약 SCM은 실현 수단의 하나로서 기법 SCM의 실시를 의무화하며, 두가지는 상반된 것이 아니고 서로 밀접한 관계를 가지고 있다.

규약 SCM과 기법 SCM은 모두 소프트웨어 개발과 보수 관리에 있어서 매우 유효한 것이다.

그러나, 소프트웨어 개발 현장에 눈을 돌려 보면, SCM의 보급 상황은 매우 부진하며 특히 국내에서는 미국의 NASA와 같은 SCM을 추진하는 구입자가 없었다는 것을 하나의 원인으로 들 수 있다. 또 하나의 원인으로는 관리자의 변경에 대한 가치관 문제이다. 즉, "사양 변경=좋지 못한 것, 수정=백트래킹"이라고 생각하여, 변경을 합리적으로 다루기 전에 변경을 해서는 안된다고 생각하는 경향이 많다는 점이다.

소프트웨어 개발과 보수에서의 관리는 대량 생산에서의 생산 관리 및 품질 관리와는 다른 것이다. 소프트웨어의 생산 형태에 있어서 변경을 없앨 수는 없기 때문에 변경을 합리적으로 다룰 필요가 있는데, 이를 위해 SCM은 중요한 기술이라고 볼 수 있다.

여기에서는 소프트웨어 매니지먼트 입장에서 SCM을 소개하고, SCM이 가장 효력을 발휘하는 보수 관리에 대해서도 간단히 설명하기로 한다.

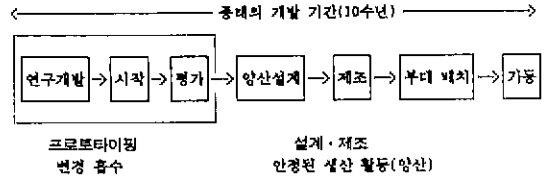
2. 규약 SCM

소프트웨어 발주자와 공급자의 양자간 계약에 의한 시스템 개발에 있어서, 개발 도중에 생기는 변경이란 번거로운 문제로서, 변경은 품질 저하, 납기와 비용 증가를 가져온다. 따라서, 변경에 의해 발생되는 이득과 악영향을 합리적으로 다루기 위해서는 공급자와 구입자 간에서의 사전 약속이 필요하다. 이러한 배경에서 생겨난 것이 규약으로서의 SCM이다.

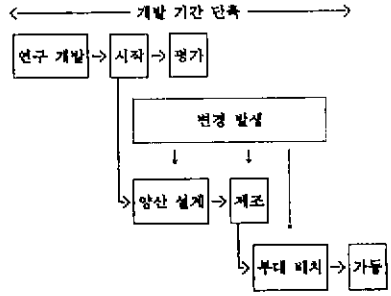
2.1 형상 관리의 역사

우선 형상 관리의 역사적 배경을 설명하기로 한다. 하드웨어를 대상으로 하는 CM(Configuration Management) 그 자체가 생겨난 것은 동서냉전 상태에 있었던 1950년대 미국에서의 병기 개발 분야이었다. 당시 사용한 병기 개발의 라이프싸이클은 (그림 2)에 나타낸 바와 같은 프로토타이핑(prototyping)형으로서, 기술적인 불안정 요소를 없애기 위해 프로토타이핑을 통해 변경이 적고 안정된 생산 활동을 하는 방식이었다. 이 방식의 문제점은 라이프싸이클이 너무 길고 복잡하다는 점으로, 한번 개발하는데 무려 10년이나 걸렸다.

이러한 방식으로는 "경쟁 상대를 추월한다"는 목표를 달성하기까지 여러 번 개발을 할 필요가



(그림 2) 프로토타이핑에 의한 개발 라이프싸이클



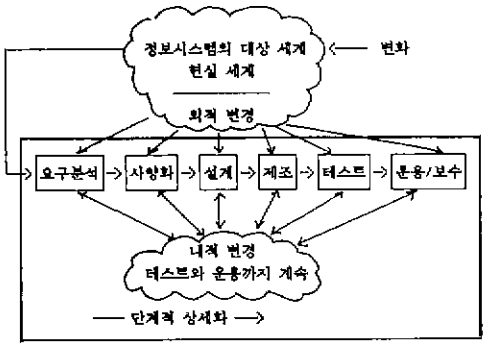
(그림 3) 형상 관리에 의한 개발 기간 단축

있기 때문에, (그림 3)에 나타낸 것과 같이 개발을 하면서 생산하고 동시에 각급 부대에 배치하는 새로운 개발 방법을 취하게 되었다. 물론 이 방식의 경우 많은 변경이 생긴다. 제품이 생산되는 동안에 변경되어 제품 하나하나가 다른 형상과 버젼을 가질수록 복잡한 상황이 생긴다.

이와 같이 만들면서 변경하는 생산 방식을 취해 온 것이 CM이다. 그후 CM은 민간 부문에도 널리 확산되어 신뢰성이 높고 불안정한 구성 요소를 가지며 개발 기간이 중요한 분야의 생산과 보수의 관리 기법으로서 특히 항공기와 컴퓨터 분야에서 널리 사용되고 있다.

2.2 소프트웨어 변경의 종류

소프트웨어에 있어서 변경은 하드웨어의 경우와 약간 상황이 다르다. (그림 4)에 나타낸 바와 같이 소프트웨어의 변경에는 생산 활동의 밖에서 생기는 "외적 변경"과 안(내부)에서 생기는 "내적 변경"의 두 종류가 있다.



(그림 4) 소프트웨어 개발에서의 변경의 종류

(1) 외적 변경

하드웨어의 CM이 대상으로 하는 변경은 기술 변경(Engineering Change)이라고 불리는 것으로서 설계시의 기술적인 불안정에 의해 생긴다. 그러나, 소프트웨어의 변경은 요구 그자체가 불안정함으로서도 생기는데, 요구 사양을 비롯해서 모든 사양의 변경과 추가가 발생한다. 변경은 개발 비용과 납기, 시스템의 신뢰성과 안정성에 커다란 영향을 미치기 때문에 공급자와 구입자가 변경을 합리적으로 다루기 위한 사전 약속이 필요하다.

(2) 내적 변경

내부에서 생기는 변경이란 소프트웨어 개발 라이프싸이클에 걸쳐 대량으로 생기는 버그 수정 등이 해당된다. 소프트웨어의 변경은 직접적인 손실, 가령 부품의 폐기라든가 분해와 같은 손실이 적어서 간단히 실시할 수가 있다. 변경에 의해 파생되는 문제는 변경전까지 행한 테스트의 성과, 즉 테스트와 디버깅에 의해 구축한 소프트웨어의 신뢰성을 한순간에 잃어버린다는 점이다. 이러한 경향은 라이프싸이클의 후반일수록 영향이 크기때문에, 변경을 매니지먼트의 대상으로 할 필요가 생긴다. 따라서, 공급자는 안전하고도 확실히 시스템을 공급하기 위해서 내부에서 생기는 변경을 파악해서 제어해야 한다.

2.3 변경 파악

변경을 파악하기 위해서는 변경되는 쪽의 실체를 파악할 필요가 있다. 이것을 SCM에서는 품목 식별(Identification of Item)이라고 한다. 그러나 실체측도 개발 라이프싸이클에 따라 사양서로부터 소스코드로 바뀌게 되기 때문에, 이것만으로는 무엇이 변경되었는지를 알 수 없다. 이를 위해서 (그림 1)의 윗부분에 나타낸 것처럼, 기준선(Base-line)과 버전 관리를 이용해서 변경 내용을 파악한다.

(1) 품목과 요소(component)

품목 식별이란 대상으로 하는 소프트웨어가 어떠한 요소로 구성되고, 무엇에 의해 그 특성과 인터페이스가 정의되어 있는지를 분명히 하는 것이다. 식별은 계층 구조로 행해지는데, 위에 있는 단위를 품목이라 하고, 품목을 구성하는 것을 요소라고 한다. 구체적인 품목에는 프로그램과 그들을 모은 서브시스템이 대응된다. 그리고 사양서와 루틴 등을 요소라고 부른다. 품목과 요소는

식별명(또는 식별 번호)에 의해 식별된다.

(2) 기준선

품목이란 개발 라이프싸이클이 진행됨에 따라 형태가 변한다. 가령 기능 사양서→구조 설계서→상세 설계서→모듈→프로그램으로 공정에 따라 품목의 형태와 형상이 크게 변한다. 라이프싸이클 상의 진행과 변경을 구별하기 위해, 라이프싸이클 상의 진행을 구분해 둔다. 이 구분을 기준선이라 부르는데, 품목을 구성하는 요소와 그 관계를 포함해서, 공식적으로 식별한 덩어리로서 기준선을 설정한다. 기준선이 설정되면 구성 요소는 제3자에 의한 형상 관리 대상이 되고, 그 이후의 수정은 개발 행위가 아니라 변경으로서 다른 절차에 의해 관리된다.

(3) 버전

버전은 기준선이 설정된 후에 행해진 변경에 대한 기록으로서, 변경이 허가되고 변경이 실시되면 버전이 바뀌게 된다. 식별명(또는 식별 번호)과 버전에 의해 시스템의 구성 요소는 명확히 식별된다.

2.4 SCM의 실시

(그림 1)의 왼쪽 아래 "형상 관리의 절차"에 나타낸 것처럼, SCM이 실시되고 있는 상황을 형상 제어(Configuration Control) 상태라고 한다.

(1) 형상 제어

SCM에서 중요한 것은 형상 제어 상태를 유지하는 것이다. 형상 제어 상태란 품목과 구성 요소가 그것을 직접 담당하는 기술자의 손을 떠나서, 관리 상태에 놓여지는 것을 의미한다. 이 형상 제어 상태로 옮겨지는 것을 동결(Freeze)이라고 부르기도 한다. 동결된 상태란 변경을 금지하는 것이 아니라, 구성 요소를 수정하기 위해 해야 할 변경 절차가 설정되고, 매니지먼트의 대상으로 하는 것이다. 동결된 후에는 기술자가 마음대로 수정할 수가 없는데, 이것은 변경을 매니지먼트하기 위한 수단이다. 형상 제어가 행해지고 있는 상태에서 변경 관리(Change Control)를 할 수 있게 된다.

(2) SCM 계획과 조직

SCM은 개발자와 보수자에게만 관계있는 것이 아니라, 구입자와 서브컨트랙터(subcontractor)등 시스템 개발에 종사하는 모든 사람과 관계가 있다. SCM을 확실히 적용하기 위해서는 프로젝트의 초기 단계에서 SCM을 행하는 역할과 절차를 정의한 SCM 계획이 필요하다. 각각의 조직과 프로젝트별로 개별적으로 SCM 계획서를 작성하는 것은 산업계로서는 커다란 손실이다. ANSI/IEEE의 규격은 SCM 절차 등을 SCM계획서로서 제공하는데, ANSI/IEEE의 SCM 계획서에는 다음 항목이 포함되어 있다.

- SCM을 실시하는 조직과 책임을 명시한다
- 형상제어위원회(Configuration Control Board : CCB)를 설정해서 변경 심사를 한다
- 구성 품목의 범위와 기준선을 정한다
- SCM부문의 업무 계획을 작성한다
- 변경의 승인에 대한 권한을 정한다
- 이용하는 기법 SCM을 정한다
- SCM에 대한 감사를 한다

등이다.

(그림 1)의 왼쪽에서 소개한 바와 같이, 형상 관리는 CCB와 SCM 부문에 의해 적용된다. SCM 부문은 동결된 품목의 형상 제어를 하고, CCB는 변경을 심사해서 변경의 승인을 한다.

3. 기법 SCM

규약으로서의 SCM에서는 개발 초기 단계에 계약 및 기준과 함께 SCM 계획이 작성되어서 적용되며, 외적 변경을 포함해서 변경을 대국적 관점에서 관리하는 기법이다.

한편, 생산 현장에 있어서 변경 문제는 사양서 단계보다도 작성된 프로그램쪽이 보다 중요하다.

3.1 프로그램 구성

프로그램은 조그만 소스 모듈로 분할해서 작성

된다. 부품인 소스 모듈을 컴파일러와 같은 개발 툴을 이용해서 오브젝트 모듈로 변환하여 최종적인 프로그램을 만든다. 개발 툴을 이용해서 소스 모듈로부터 오브젝트 모듈과 로드 모듈을 만들어 내는 것을 도출(Derivation)이라고 하는데, 도출 과정은 조립 과정이기도 하다. 프로그램의 구성 요소와 도출 관계를 (그림 5)에 나타냈는데, 여러 종류가 있다.

초기의 소프트웨어 개발에 있어서는 이 조립 과정을 작업자가 개별적으로 했지만, 작업자의 수가 증가하면서 여러 가지 문제가 발생하게 되었다. 이러한 문제 중에서 대표적인 문제가 수정했다고 여기고 있던 것이 프로그램에는 전혀 반영되지 않고 전의 상태대로 남아 있는 것이다[8]. 이 문제를 해결하기 위해 툴이 만들어지고, 툴을 토대로 하여 형상 관리 절차가 적용되게 되었다. (그림 5)에 나타낸 구성 요소에는 다음과 같은 종류가 있다.

(1) 기술 문서

개발 단계에서 만들어진 여러 가지 기술 문서가 포함된다. 컴퓨터 기억 장치에 들어 있는 문장(텍스트 화일)도 툴에 의한 형상 관리의 대상에 포함된다.

(2) 소스 코드

프로그램 언어로 기술된 소스 모듈 이외에도 여러 형태의 소스 코드가 있다. 가령, 컴파일러에 대한 옵션이라든가, 데이타 구조체를 정의하는 매크로 소스 코드 등이 있다.

(3) 개발 툴

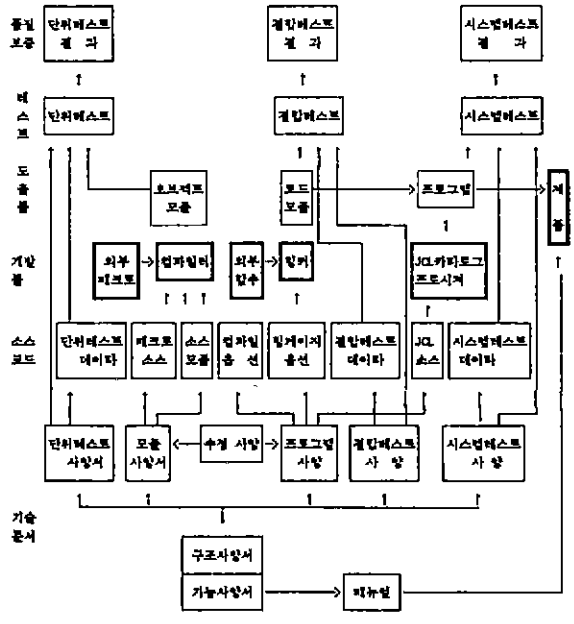
개발 툴 자신과 그 버전도 프로그램의 중요한 구성 요소이다. 개발 툴에는 컴파일시에 인라인 전개되는 매크로와 함수 라이브러리 등도 포함된다.

(4) 도출물

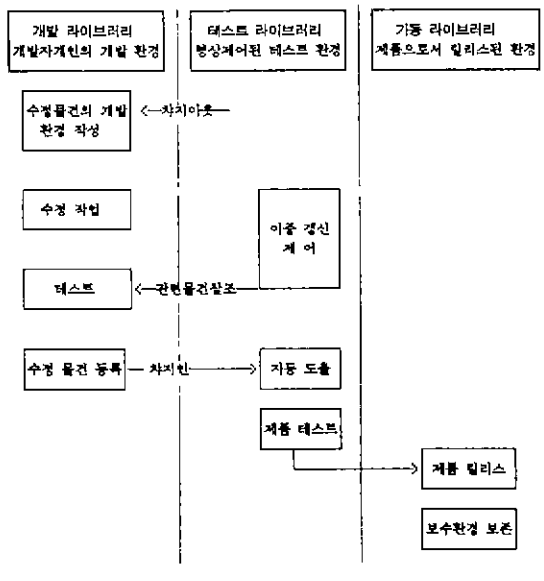
소스 코드를 토대로 하여 개발 툴에 의해 만들어진 것이 도출물로서, 도출물은 대응하는 소스 코드의 버전과 대응되어야 한다.

(5) 테스트와 품질 보증

테스트 데이타와 테스트 환경 및 테스트 결과도 구성 요소이다. 변경이 생기면 그와 동시에 테스트를 해야 하며, 테스트 결과도 식별되어 버전 관리의 대상이 된다.



(그림 5) 프로그램의 구성 예



(그림 6) 라이브러리 구조

3.2 프로그램 수정 절차

라이브러리는 (그림 6)에 나타낸 것과 같이 통상 3단계로 구성되며, 이들은 개발 라이브러리, 테스트 라이브러리, 가동 라이브러리라고 불리는데, 테스트 라이브러리와 가동 라이브러리가 형상 제어를 받는다. 개발 라이브러리는 개발자 개인용의 개발 환경을 제공하는 것으로서, 동시에 행해지는 다른 개발자와 경합하지 않도록 설정된다.

(1) 수정 허가와 챠지아웃(charge-out)

필요한 변경이 허가되면 담당자는 해당 구성 요소(소스 모듈 등)를 테스트 라이브러리에서 가지고와서 개발 라이브러리에 복사한다. 그후 수정이 끝날 때까지 다른 개발자가 이중으로 수정을 하지 못하도록 구성 요소를 라이브러리에서 복사하지 못하게 하는데, 이러한 것을 챠지아웃이라고 한다.

(2) 도출과 테스트

해야 할 수정을 하고, 수정에 의해 재도출이 필요한 것을 일시적으로 개발 라이브러리상에 재도출한다. 이것은 매크로를 수정한 경우에 그것을 인용하고 있는 소스 모듈을 재컴파일하는 것에 해당한다. 도출한 오브젝트를 테스트해서 수정 확인을 한다.

(3) 차지인(charge-in)과 조립

개발 라이브러리 상에서 테스트에 합격하면 테스트 라이브러리에 다시 넣는데, 이것을 차지인이라고 한다. 챠지인은 수정한 구성 요소에 대해서만 한다. 그 수정에 의해 파생되는 재도출은 테스트 라이브러리 상에서 행해지고, 그 요소를 포함하여 베이스 라인이 갱신된다.

(4) 가동 라이브러리로 승격

테스트 라이브러리 상에서는 다른 변경과 동기를 취해서 제품 레벨의 테스트가 행해진다. 테스트 결과가 공식적으로 승인되면 테스트 라이브러리로부터 가동 라이브러리로 복사된다. 매뉴얼 등이 첨부된 구성 품목도 포함하여 제품 베이스 라인이 설정된다.

3.3 대표적 도구

기법 SCM을 지원해 주는 것이 형상 관리 툴로서, 오래전부터 여러가지 툴이 제안되어 사용되고 있다.

(1) make [8, 9, 10]

소스 모듈을 수정했다면 그 소스로부터 도출되는 오브젝트 모듈과 로드 모듈을 재작성할 필요가 생긴다. 수정된 소스 모듈이 프로그램 간에서 공용되는 레코드 정의이었다면 재도출해야 할 범위는 복잡하고 영향이 광범위하게 미치게 된다. 재도출이 필요한 모듈만을 사람손으로 골라낸다는 것은 어려운 일인데, make 툴은 이 문제를 자동적으로 해결해 준다. 수정을 했을 때, make-file에 미리 기술된 도출 관계와 도출해야 할 화일의 수정 일시(타임 스탬프)를 토대로 해서 필요한 요소만을 재컴파일 또는 재링크한다.

(2) SCCS/RCS [8, 9, 14, 22]

소스 모듈의 형상 관리는 라이브러리에 대해서 챠지인/챠지아웃을 함으로써 실현된다. 이 대표적인 툴로서 SCCS(Source Code Control System)과 RCS(Revision Control System)이 잘 알려져 있다. 이들 툴은 프로그램만이 아니라, 도큐멘트와 데이타도 대상으로 한다. 도출되는 오브젝트와 로드 모듈은 대상으로 하지 않고, make를 기동함으로써 해결된다.

SCCS와 RCS는 세대 관리 기능을 가지고 있으며, 원본이 되는 세대 정보와 이 세대와의 차분 정보에 따라 라이브러리의 기억 효율을 향상시킨다. 각각의 버전에 대한 경력 정보를 가지고 있고, 체크 아웃후의 이중 갱신 록과 버전의 분기 기능도 가지고 있다.

(3) LifeLine/LifeStudio [11, 12]

SCCS와 make를 이용한 형상 관리의 결점으로서 여러 작성자에 의한 동시 갱신과 임의의 버전을 도출할 수 없다는 점을 들 수 있다. 이 문제를 해결하기 위해서 컴포지션이라는 구성 정보 자체를 기억해서 버전 관리를 한 것이 LifeLine이다. 이것에 의해 여러 작업자가 동시에 동일

요소를 변경할 수 있다.

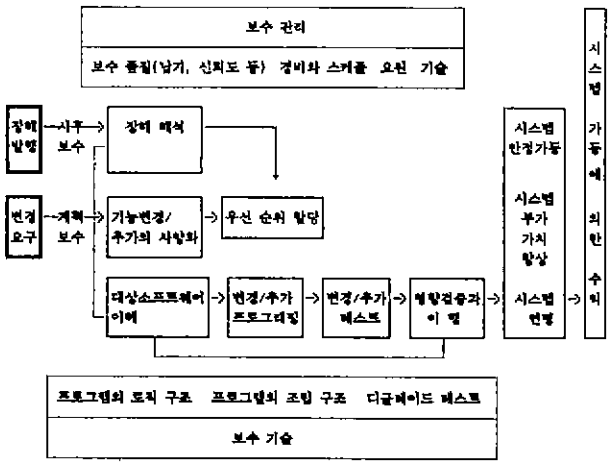
(4) Ada언어 시스템 [8, 9, 13]

Ada 언어 시스템(ALS : Ada Language System)은 미국 국방성이 프로그램 언어 Ada를 도입하기 위해서 개발한 구성 관리 기능을 포함하는 개발 환경이다. Ada에 의한 프로그램은 프로그램의 인터페이스를 기술하는 패키지 사양(Package Specification)과 처리를 기술하는 패키지 본체(Package Body)로 나누어진다. Ada 컴파일러는 관계되는 패키지 사양간의 엄밀한 체크를 하는 기능을 가지고 있다. Ada 언어 시스템은 화일 구조를 가지고 있으며 디렉토리 구조와 세대 관리 기능도 포함하고 있다. 이 화일 시스템의 특징은 화일에 대한 어소시에이션(association)이라고 하는 화일간의 관계 정보를 가지고 있다는 것이다.

4. 보수 관리

소프트웨어 보수의 목적은 시스템의 안정된 가동과 부가 가치의 향상 또는 시스템의 연명을 통해 시스템 가동에 의해 얻어지는 가치를 극대화하는 것이다.

하드웨어와 달리, 가동중인 시스템을 변경해서 시스템 가치를 향상시킬 수 있다는 것은 소프트웨어의 커다란 특징이다. 그러나, 조그만 보수 상의 에러로 인해 커다란 손실이 생길 위험성도 있다.



(그림 7) 보수 작업의 흐름

소프트웨어의 보수 작업은 매니지먼트 면에서나 기술 면에서나 까다로운 문제이다. 보수 작업의 흐름을 단순화한 것을 (그림 7)에 나타낸다. 보수 작업은 크게 두 가지로 나눌 수 있는데, 하나는 "사후 보수"라고 불리는 것으로서 발생한 장애에 대응하는 작업이다. 또 하나는 "계획 보수"라고 불리는 것으로서, 변경 요구에 의해 변경과 추가를 계획적으로 하는 일종의 개발 작업이다.

4.1 보수에 필요한 지식

보수가 신규 개발과 비교해서 까다로운 이유는 대상이 되는 소프트웨어를 미리 이해해 두어야 한다는 점이다. 보수를 하는데 필요한 지식으로는 다음의 세가지를 들 수 있다.

(1) 프로그램의 로직 구조

보수의 대상이 되는 프로그램 기능과 내부 구조 및 논리를 이해해야 한다.

(2) 프로그램의 조립 구조

(그림 5)에 나타낸 바와 같이, 프로그램은 소스 코드 이외에 여러 가지 구성 요소로 되어 있는데, 이들 구성 요소도 보수에 영향을 미친다. 가령, 사용한 컴파일 옵션이 불분명하다거나, 이용한 컴파일러(개발 툴)의 버전이 불분명할 경우, 소스 프로그램을 수정했다고 하더라도 제대로 작동되지 않는다.

과거에는 프로그램 수정을 패치라고 하는 프로그램 기계어를 직접 수정하는 방법이 이용되었다. 이 방법은 프로그램을 재조립하지 않는 보수 방법으로서, 이때문에 수정에는 많은 시간이 소요되지만 조립에 의한 에러는 생기지 않는다.

오늘날은 보수 작업을 패치로 하는 경우는 극히 드물며, 소스 코드 레벨에서 보수를 하기 위해 재조립의 환경을 보존해 두고, 그것을 이해하는 것이 필수적이다.

(3) 프로그램의 테스트 구조

보수에 소요되는 비용중에서 테스트, 그 중에서도 수정이 다른 곳에 악영향을 미칠지 여부의 테스트가 많은 비중을 차지하고 있다. 이 테스트를 테스트 유용으로 실시할 것인지, 또는 수정할 때마다 다시 테스트 설계부터 행할지는 커다란 차이이다.

테스트를 유용하는데는 테스트 환경과 이전의 테스트 결과를 보존해 두고 그것을 이해할 필요가 있다.

4.2 보수에서의 형상 관리의 역할

보수는 개발 성과에 의존한다는 것은 분명하다. 개발 성과물 중에서 어떤 것이 보수에 필요할까? 직감적으로는 에러가 적고 이해하기 쉬운 프로그램으로서, 코멘트의 삽입등 프로그램의 로직 구조라고 생각되지만, 프로그램의 조립 구조와 프로그램의 테스트 구조도 중요하다. 프로그램의 조립 구조와 프로그램의 테스트 구조의 문제는 형상 관리가 확실히 실시되지 않으면 해결되지 않는다. 확실히 실시된다는 것은 매니지먼트 하에서 SCM이 계획되고, 역할 분담이 이루어지며 컨트롤되어야 함을 의미한다.

개발과 보수는 동일한 조직이 행한다고는 할 수 없다. 외부에 발주한 소프트웨어의 경우, 구입자가 보수를 해야 하는 것이 현실적이다. 그렇게 되면 보수에 있어서 형상 관리는 중요한 역할을 한다. 개발시의 형상 관리 방법이 다양한 것은 구입자에게 있어서 커다란 손실을 가져온다. 따라서, ANSI/IEEE의 표준과 ISO/IEC의 국제 표준으로서 제정될 필요성이 있다고 본다.

5. 결 론

소프트웨어 형상 관리에 대한 설명대로 SCM은 변경을 하지 못하도록 하는 것이 아니라, 변경을 합리적으로 다루기 위한 관리 기법이다.

매니지먼트 입장에서 본 형상 관리의 의의는 개발자와 담당 산출물의 분리, 제3자에 의한 성과물의 체계적 관리, 개발 행위와 변경 행위를 분리하는 것이다. 그리고, 개발 행위에 대한 관리 이상으로 변경에 대해 중점을 두고 관리를 하는 것이다.

그를 위한 구체적인 방법으로서 관계자 간의 절차로서의 기능(규약 형상 관리)과, 프로그램 개발에 있어서 프로덕트 관리 기능(기법 형상 관리)이 있다.

SCM은 소프트웨어에 있어서 중요한 문제, 가령 소프트웨어의 구매와 계약에 있어서 변경 취급, 분산 개발 환경에 있어서 통제 문제, 보수 문제 등의 해결에 없어서는 안되는 것이다. 그러나, SCM이 소프트웨어 매니지먼트의 기반으로서 정착되기 위해서는 아직도 많은 문제가 있다. 이 분야에 있어서 현장에서의 활발한 전개를 기대한다.

참 고 문 헌

1. 森口繁一編 : ソフトウェア品質管理, 日本規格協會(1990).
2. GE編 井上義裕他譯 : ソフトウェア工學ハンドブック, マグロウヒル(1988).
3. ANSI/IEEE Std 828, IEEE Standard for Software Configuration Management Plans, IEEE (1984, 1990).
   東 基衛監修, ANSI/IEEE ソフトウェア規格集, 日本規格協會(1988).
4. ANSI/IEEE Std 1042, IEEE Guide to Software Configuration Management, IEEE (1988).
   東 基衛監修, IEEE ソフトウェア規格集 第2集, 日本規格協會(1991).
5. Dod 5200, 28-STD, Department of Defence Trusted Computer System Evaluation Criteria, Office of Standards and Products, Maryland(1985).
6. 飯塚悦功編 : ソフトウェアの品質保證, 日本規

格協會(1990).
7. Parikh, G. : Handbook of Software Maintenance, John Wiley & Sons. Inc(1986).
テクノピック譯, ソフトウェア・メンテナンス・ハンドブック, 啓學出版(1989).
8. Babich, W. A. : Software Configuration Management Coordination for Team Productivity, Addison-Wesley, New York(1986).
菊池豊彦譯, プログラムのチ-ム開發入門, CQ出版社(1988).
9. Whitgift, D. : Methods and Tools for Software Configuration Management, John Wiley, Chichester(1991).
10. Feldman, S. : Make-A Program for Maintaining Computer Programs, Software Practice and Experience, 9(3), pp.255-265(1990).
11. 坪谷英昭, 岸 知二他 : 構成・版管理ライブラリLifeLineを用いたC言語プログラム開發環境, 情報處理學會, ソフトウェア工學研究會, SE-73(14), pp. 107-114(1990).
12. 池田健次郎, 岸 知二他 : 共同開發時におけるインテグレ-ション支援機能, 情報處理學會, ソフトウェア工學研究會, SE-79(3), pp. 1-8 (1991).
13. Booch, G. : Software Engineering with Ada, Benjamin/Cummings, Redwood City, CA (1987).
14. Rochkind, M. J. : The Source Code Control System, IEEE Transactions on Software Engineering, SE-1(4), pp.365-370(1975).
15. Knudsen, D. B., Barofsky, A. and Satz, L. R. : A Modification Request Control System, Proceeding of the 2nd International Conference on Software Engineering, pp.187-192, IEEE and ACM, New York(1977).
16. Bersoff, E. H., Henderson, V. D. and Siegel, S. G. : Principles of Software Configuration Management, Prentice-Hall, Englewood Cliffs, NM(1979).
17. Lampson, B. W. and Schmidt, E. E. : Organizing Software in a Distributed Environment, Proceeding of the SIGPLAN 83 Symposium on Programming Language Issues in Software Systems, San Francisco(1983).
18. ANSI and AJPO, Military Standare, Ada programming Language, ANSI/MIL-SD-1815A(1983).
19. Kruskal, V. : Managing Multi-Version Programs with Editor, IBM Journal of Research and Development, 28(1), pp.74-81(1984).
20. Bersoff, E. H. : Elements of Software Configuration Management, IEEE Transaction on Software Engineering, Jan. 1984, pp.79-87 (1984)
21. Goldfine, A. and Konig, P. : A Technical Overview of the Information Resource Dictionary System, Computer and Standards, 1, pp.153-208(1985).
22. Tichy, W. F. : RCS-A System for Version Control, Software-Practice and Experience, 15(7), pp.637-654(1985).
23. Tichy, W. F. : Smart Recompilatiom, ACM Transactions on Programming Languages and Systems, 8(3), pp.273-291(1986).
24. Dowson, M. : Integrated Project Support with IStar, IEEE Software 4(6), pp.6-15(1987).
25. Lablang, D. B. and Chase, R. P. : Parallel Software Configuration Management in a Network Environment, IEEE Software, 4(6), pp.28-35(1987).
26. Tichy, W. F. : Tools for Software Configuration Management, Proceeding of the International Workshop on Software Version and Configuration Control, pp.1-20, Teubner Verlag, Stuttgart(1988).
27. Reps, T., Horwitz, S. and Prins, J. : Support for Integrating Program Variants in an Environment for Programming in the Large, Pro-

ceeding of the International Workshop on Software Version and Configuration Control, pp. 197-216, Teubner, Stuttgart(1988).

28. Clemm, G. M. : The Odie Specification Language, Proceedings of the International Workshop on Software Version and Configuration Control, pp.144-158(1988).
29. Montes, J. and Haque, T. : A Configuration Management System and More!, Proceeding of the International Workshop on Software Version and Configuration Control, pp.217-227, Teubner Verlag, Stuttgart(1988).
30. Winkler, J. F. H. and Stoffel, C. : Program Variations-in-the Small, Proceeding of the International Workshop on Software Version and Configuration Control, pp.175-196, Teubner Verlag, Stuttgart(1988).
31. Sarnak, N., Bernestein, R. and Kruskal, V. : Creation and Maintenance of Multiple Versions, Proceeding of the International Workshop on Software Version and Configuration Control, pp.264-275, Teubner Verlag, Stuttgart(1988).

양 해 술

1975년 홍익대학교 공과 대학 전기공학과 졸업(학사)
1978년 성균관대학교 정보처리학과 정보처리 전공(석사)
1991년 日本 오사카대학 기초공학부 정보공학과 소프트웨어공학 전공(공학박사)
1975년~79년 육군중앙경리단 전자계산실 시스템분석장교 근무
1984년~92년 성균관대학교 경영대학원 강사
1986년~87년 日本 오사카대학 객원연구원
1993년~94년 한국정보과학회 학회지 편집부위원장
1980년~95.5 강원대학교 전자계산학과 교수
1994년~현재 한국산업표준원(IIS) 이사
1994년~현재 한국정보처리학회 논문지편집위원장
관심분야 : 소프트웨어 공학(특히, S/W 품질보증과 평가, SA/SD, OOA/OOD/OOP, CASE, SI), 소프트웨어 프로젝트관리.

박 정 호

1980년 성균관대학교 사범대학 졸업(문학사)
1980년~82년 성균관대학교 경영대학원 정보처리 학과(경영학석사)
1985년~87년 日本 오사까대학교 대학원 정보공학전공(공학석사)
1987년~90년 日本 오사까대학교 대학원 정보공학전공(공학박사)
1994년~현재 한국정보처리학회 학회지 편집위원장
1991년~현재 선문대학교 전자계산학과 교수
관심분야 : 분산 알고리즘, 소프트웨어공학, 시스템통합.