□특집□

소프트웨어 유지보수의 자동화

유 홍 준†

◈ 목 차 ◈

1. 서 론

소프트웨어 유지보수(software maintenance)는 '소프트웨어의 출하 후의 결함의 제거, 성능이라던가 다른 속성의 향상 또는 환경의 변화에 적응하기 위한 수정'이라고 정의할 수 있다(ANSI/IEEE규격 729-1983). 소프트웨어 유지보수 자동화(SMA: Software Maintenance Automation)라 함은 소프트웨어 유지보수의 제반 작업단계를 자동화 함을 뜻한다 [20].

소프트웨어 유지보수는 수십 스탭(LOC: Lines Of Code)이내정도의 규모의 프로그램이 하나의 파일로 구성되는 식의 단순한 것이라면, 별도의 도구라던가 방법론없이도 프로그래머 개인의 능력에 의해 충분히 해결될 수 있다.

하지만, 규모가 커져서 수십만~수백만 스탭에 이르는 방대한 량이 된다면 이야기는 달라진다.

소프트웨어는 자체의 불가시성과 가변성, 무한 확장성 등으로 인해 인간의 이해능력을 초과하는 복잡성을 가지고 있다(Fred Brooks, North Calorina University). 또한 인간이 폐기시키지 않는 한 그 수명이 영원히 지속되는 영생성으로 인해 일단 개발이 완료된 후에는 지속적으로 유지보수가 이루어져야 하는 특성을 가지고 있다. 인간을 하드웨어 적인 측면에서, 출생전 10개월동안을 개발기간으로 출생이후의 수십년을 유지보수 기간으로 볼 수 있지만, 정신적인 측면에서는, 어떤 인간이 한 번 확립해 놓은 사상적 기반이 인간의 육체라는 하드웨어가 소멸한 후에도 후세에까지 계속 많은 영향을 미치며 수정 보수되는 것과 마찬가지로, 소프트웨어의 유지보수는 개발에 비해서 많은 비율을 점유할 수밖에 없는 것이다.

하드웨어의 경우에는 하드웨어의 사양을 완벽하게 담고 있는 사용자 매뉴얼(user's manual)이나 기술 매뉴얼(technical manual) 등에서 트러블 슈팅(trouble shooting)까지 정확하게 할 수 있도록 보수 지침이 만들어져 있기 때문에, 보수작업이 아주 쉽게 이루어질 수 있다. 또한 보수작업이 이루어지는 과정에도 사양이 변화되지 않는다. 하지만, 소프트웨어의 경우에는 개발시에 만들어진

† 종신회원 : 힘스미디어 대표

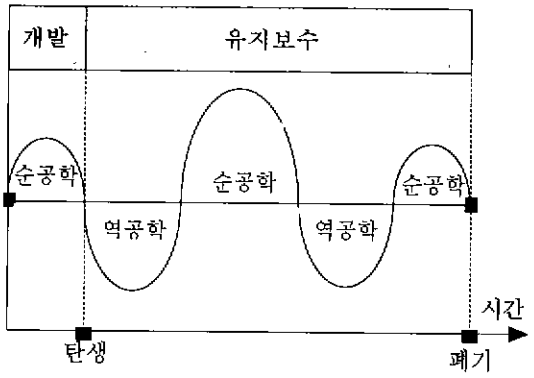
사양서는 그대로 있는데, 코드의 경우에는 보수하는 과정에서 사양이 점점 변화되기 때문에 일정한 시간이 지난뒤에는 비치된 사양서와 실제의 소프트웨어 코드 사양간에 현저한 차이가 발생하여 보수가 점점 어렵게 되는 특징을 가지고 있다.

결국 소프트웨어에서 보수를 쉽고 정확하게 행하기 위해서는 보수작업의 전공정에서 자동화의 비율을 높혀 인간의 개입을 최소화하고, 보수작업이 행해짐과 동시에 사양서의 업데이트가 동시에 이루어지게 하는 것이 바람직하다.

소프트웨어 유지보수 작업의 자동화, 즉 SMA (Software Maintenance Automation) 가 필요한 이유가 여기에 있다.

2. SMA 지원기술의 분류

소프트웨어의 보수(maintenance) 공정을 프로그래머의 개인적인 능력에 의존하지않고 공학적인 입장에서 자동화시키기 위해서 필요한 소프트웨어 유지보수 자동화(SMA: Software Maintenance Automation)를 지원하는 기본 기술을 시간축을 중심으로 파악하여 나타내면 (그림 1)과 같다 [29, 30, 31, 32, 33].



(그림 1) SMA를 지원하는 기본 기술

소프트웨어는 개발이 완료되어 정식으로 목적에 맞는 일을 수행할 수 있도록 탄생하여, 필요성이 소멸되어 폐기될 때까지의 기간동안 끊임없이 유지보수 작업이 이루어지게 된다.

소프트웨어 유지보수와 관련한 기술은 <표 1>과 같이 크게 3가지로 구분된다 [22].

<표 1> 소프트웨어 유지보수 관련 기술의 구분

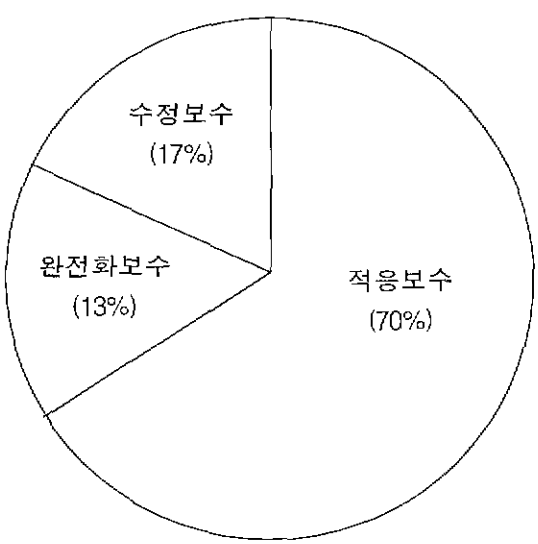
구 분	내 역
보수지향기술	요구정의, 설계, 코딩과 디버그, 테스트
보수지원기술	정보수집, 버그원인해석, 프로그램 수정과 제품관리, 수정확인, 보수용문서화
예방보수기술	부하특성수집, 운전상황관리, 운용품질관리

이러한 유지보수관련 기술을 적용하기 위한 동기가 되는 원인은 <표 2>와 같이 크게 3가지의 기본 종류로 파악할 수 있다(Swanson).

<표 2> 유지보수관련 기술 적용을 위한 동기

구 분	내 역
결함	- 소프트웨어내에 포함되어 있는 에러로 인해, 프로그램이 비정상적으로 종료하거나, 출력 결과가 틀리게 되는 등의 예
환경변화	- 데이타베이스의 재구축 등과 같은 데이타 환경의 변화, 새로운 운영체제의 도입이라던가 새로운 하드웨어 플랫폼으로의 프로그램 의이행 등과 같은 처리환경의 변화의 예
요구	- 개발자에 대한 사용자 또는 소프트웨어 보수담당자로부터의 프로그램 조작방법의 개선, 화면구성의 변경, 새로운 기능의 추가 등과 같은 요구의 예

이러한 원인에 대한 보수활동(maintenance activity) 으로서, 소프트웨어의 결함에 대한 수정보수(Corrective Maintenance), 환경의 변화에 대한 적응보수(Adaptive Maintenance), 사용자라던가 보수담당자의 요구에 대한 완전화보수(Perfective Maintenance)의 3가지 보수기술이 정의된다. 최근에는 여기에다 예방보수(Preventive Maintenance)를 추가하여 보는 경향이 있다. 우선 기본적인 입장에서 3가지로 분류하여 파악한다면, 이들 3가지의 보수기술의 각각이 전체보수에 있어서 점유하는 비율은 1989년의 George Dinardo의 조사에 의하면 (그림 2)와 같이 환경의 변화에 적응하기 위한 적응보수(adaptive maintenance)가 차지하는 비율이 급격히 증가하고 있어서, 다른 어떠한 보수기술보다도 중요한 비중을 차지하고 있음을 알 수 있다(George DiNardo의 조사, 1989) [22].



(그림 2) 보수작업의 점유비율(출처: 1989년 DiNardo의 조사)

그런데, 현실은 어떠한가? 급속하게 변화하는 시스템 환경속에서 유지보수의 실마리를 찾지 못하고 있다. 특히 개발 및 보수요원의 이직, 전직, 전보발령 등과 같은 원인에 의한 잦은 보수요원의 교체로 인해 새롭게 투입된 보수요원은 환경의 변화에 신속하게 변화된 보수를 수행하기는 커녕, 오히려 기존의 개발된 소프트웨어의 내용을 이해(understand)하는 데에만 전체 보수작업의 47%이상을 소모해 버리고 있다는 사실이 IBM사의 Fjeldstad와 Hamlen의 조사에 의하여 밝혀졌다 [22].

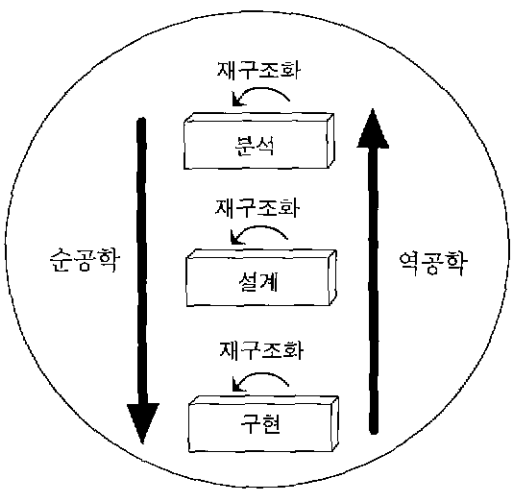
소프트웨어가 애당초 개발될 당시부터 이해하기쉽도록 만들어진다면 이러한 소프트웨어 내용의 이해에 소모되는 엄청난 시간적·물질적·인적인 손실을 최대한 절약하여 실제적인 보수작업에 투입할 수 있게 됨으로써 유지보수 작업의 능률을 극대화 할 수 있게 됨은 당연한 이치이다. 이처럼 소프트웨어를 이해가 용이하도록 각 단계별로 정확한 문서를 생성하고, 간 단계간의 수직적인 측면에서의 이행을 자동화하기 위한 기술이 순공학(forward engineering)과 역공학(reverse engineering) 기술이다. 수평적인 측면에서는 각 단계별로 재구조화(restructuring) 작업의 투입이 필요하다. 재구조화 작업은 순공학 및 역공학을 적용하기 위한 전단계로서 필요에 따라 적절하게 적용되어야 한다.

이처럼 수평과 수직을 모두 감안한 S/W 개발·유지보수 관련 용어를 1991년의 GUIDE AE-3100 Reverse Engineering Report에 의거하여 정의하면 <표 3>과 같다 [20].

소프트웨어 유지보수의 자동화를 지원하기 위하여 꼭 필요한 기반기술을 시각적으로 나타낸 것이 (그림 3)이다.

<표 3> S/W 개발 · 유지보수 관련 용어

구 분	내 역
순 공 학	- 실현에 의존하지 않는 고수준의 추상적 개념으로부터 실현에 의존하는 저수준 구체적 개념 표현으로의 변형. - 고수준추상적 개념 → 물리적 실현 - 요구분석 → 설계 → 구현
역 공 학	- 실현에 의존하는 저수준 구체적 개념으로부터 실현에 의존하지 않는 고수준 추상적 개념으로의 데이터의 기술과 프로그램 논리의 추출, 표준화, 문서화 등. - 구현 → 설계 → 요구분석 - 시스템구성요소와 관련의 해명 → 다른 형태나 고수준의 시스템 표현 - 재문서화(re-documentation) - 설계복원(design recovery) - 함수추상화(function abstraction) - 업무규칙(business rule)의 추출 · 표현
재구조화	- 코드의 구조를 개량하기 위해, 프로그램의 기능을 변경하지 않은 상태에서의 소스 코드의 재편성. - 어느 표현으로부터 다른 표현으로의 변형(추상화의 수준은 같다).



(그림 3) 유지보수의 자동화에 필수적인 기반기술

이외에도 어플리케이션을 갱신하거나 강화판을 생성하기 위해 어플리케이션을 부분적으로 수정(필요에 따라서는, 어느정도의 구조의 수정도 포함)하는 것을 재개발(re-development)이라 하기도 하지만, 소프트웨어 유지보수성(maintainability)의 증진을 지원하는 자동화 기반기술은 순공학(forward engineering), 역공학(reverse engineering) 및 재구조화(restructuring)의 3가지를 기준으로 생각하면 되며, 이를 복합적으로 재공학(reengineering) 이라는 시각으로 바라다 볼 수 있다.

재공학(reengineering)의 정의를 내려보면 <표 4>와 같다(GUIDE AE-3100 Reverse Engineering Report, 1991) [20, 22, 31, 32, 33].

<표 4> 재공학의 정의

용 어	정 의
재공학	- 새롭게 하는 것 - 기존 시스템의 분석·이해 → 새로운 형태로 변경 → 구현 - 역공학 + (순공학 | 재구조화)

재공학(reengineering)이란 기존의 소프트웨어에 대한 역공학 및 재구조화 등의 기술처리를 통해서 소프트웨어(software) 부품을 추출해 내고, 이를 다시 새롭게 순공학(forward engineering)으로 구현함으로써 재사용부품(reusable component)으로 만들어 내는데 핵심적인 역할을 하는 기술이라고 말할 수 있다. 또한 재공학 기술의 전반적인 자동화야 말로 유지보수성을 획기적으로 증진시켜줄 수 있다.

3. SMA 구현시 고려사항

소프트웨어 유지보수 자동화(SMA)를 위한 재공학 기술중에서도 가장 중요한 기술은 역공학 기술이다. 역공학 기술의 적용시에는 <표 5>와 같이 두가지 작업을 생각할 수 있다.

<표 5> 역공학 작업의 구분

작업구분	내 역
재문서화 (re-documentation)	상대적인 추상화 수준은 동일하며 의미론적으로 등가인 표현의 작성 또는 개정
설계복원 (design recovery)	소스코드, 설계문서, 개인적경험, 문제나 어플리케이션 영역에 관한 일반적인 지식의 조합으로부터, 설계의 추상적개념을 재생하는 것

역공학을 기준으로 하는 재공학에서의 문서화의 목적은 인간이 이해하기 쉽게하는 것이므로 거기에는 인간의 신체적인 인식능력에 맞는 문서화 작업의 수행이 필요하다.

재공학 전반의 작업에서 문서화하는 작업을 통해 소프트웨어 유지보수 자동화를 구현함에 있어서는 여러 가지 인간의 인지능력을 감안한 자동화 기술 적용이 필수적인데, 그 중에서도 특히 지각체제화(perceptual organization), 공간적사고(spatial thinking), 논리체계화(logical systematization), 시공간지역성(temporal & spatial locality) 및 경계인식성(boundary recognition) 등의 원리의 적용이 적극적으로 고려되어야 한다 [31, 33].

3.1 지각체제화(perceptual organization)

주의집중을 파도(wave)에 비교하여 활성화되는 인지단위와 주변의 억제영역으로 파악할 때, 우리 인간은 한 번에 고작 한 두개의 사건에만 주의를 기울여서 지각할 수 있다(Titchener).

이러한 지각능력의 한계로 인해, 인간에게 한꺼번에 여러가지의 사건이 제시되면, 두뇌의 정보처리 활동에 심각한 병목현상(bottle neck)이 일어나게 되어, 지각능력(perceptual capability)이 급격히 떨어지며 지각속도(perceptual speed) 또한 현저하게 저하된다.

인간의 지각특성을 몇가지만 열거하면 <표 6>과 같다(Max Wertheimer).

우리 인간은 관심이 미치는 부분을 좀 더 주의집중해서 도형(figure)으로 지각하며, 상대적으로 관심이 덜 미치는 부분을 배경(ground)으로 지각하는 특성을 가지고 있다. 가역적 도형-배경 형태(reversible figure-ground pattern)의 경우에 인식에 혼란을 일으키는 경우가 많은 이유가 바로 이러한 인간의 인식 특성때문이다.

도형-배경형태 지각의 경험이 인간의 형태인식(pattern recognition)의 기본이 된다.

시선이 향하는 범위에 형태(pattern)들이 흩어져

<표 6> 인간의 지각 특성

지각의 구분		내 역
형태지각 (pattern perception)	도형-배경 분리	특징적으로 응집되어 파악되는 부분이 도형(figure)이 되고, 나머지 부분이 배경(ground)이 되는 성질
지각집단화 (perceptual grouping)	근접 (proximity)	두 형태가 가까워지면 집단화되어 지각되는 성질
	유사 (similarity)	유사한 모양이나 색깔 등은 집단화되어 지각되는 성질
	순연 (good continuation)	꺽이는 쪽보다는 부드럽게 연결되는 쪽이 집단화되어 지각되는 성질
	폐쇄 (closure)	불연속부분이 보충될 경우 보다 단순하고, 규칙적이며, 대칭적이고, 연속적인 좋은 형태(good form)로 된다면, 폐쇄적으로 집단화되어 지각되는 성질

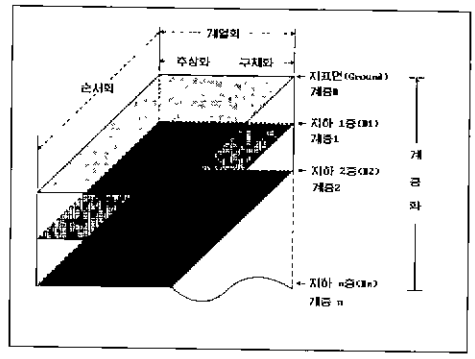
서 존재하게 되면, 서로 가까이에 있는 형태끼리 집단화되고(근접성의 원리), 비슷한 형태들로 함께 집단화되며(유사성의 원리). 파악되기 쉬운 인상적인 형태로 폐쇄적으로 집단화되어(폐쇄성의 원리) 지각된다.

이처럼, 사물이 지각될 때 하나의 응집된 형태로 체제화되는 것을 지각체제화(perceptual organization)의 원리라고 한다.

이와같은 인간의 지각 특성은 재공학 전공정의 자동화 구현시 적극 감안되어야 한다.

3.2 공간적사고(spatial thinking)

인간은 시각적인 정보가 망막에 연결된 시신경을 거쳐 대뇌에 도달하면 이것을 기억해 두었다가 언제라도 심상(心象)으로 떠올려 볼 수 있는 능력을 가지고 있다. 이렇게 영상기억(iconic memory)을 떠올리는 것을 심상주사(image scanning)라고 한다. 이때 단순히 영상만이 떠올려지는 것이 아니며, 영상과 함께 저장해 두었던 개념기억(conceptual memory) 정보도 함께 시계열적인 순위를 가지고 심상으로 떠올려진다. 심상을 이용하여 공간적인 문제를 해결하는 것을 공간적 사고(spatial thinking)라고 한다. 심상을 이용한 공간적 사고는 시계열적인 순위에 따라 4차원적으로 적용되어 문제를 해결해 나간다. 이러한 인간의 공간적 사고의 습성에 부응하는 쪽으로 만들어진 문제해결 구조가 바로 (그림 4)와 같은 잔디구조(lawn structure)이다 [30, 31, 32, 33]



(그림 4) 잔디구조의 형태

아무리 큰 프로그램이라 할지라도 잔디구조를

이용하면 간단하게 파악이 된다. 왜냐하면 실제의 유지보수에서, 2차원 평면상에 그려진 것이라 할지라도 심상(心象)을 통해 공간적 시각으로 보고 느껴서 파악할 수 있도록 하기 때문이다.

특히, 유지보수의 자동화를 구현함에 있어서는 유지보수를 위한 틀(frame)이 필요하게 되는데, 그 때에 사용되는 인공구조가 바로 잔디구조이다.

3.3 논리체계화(logical systematization)

논리는 일정한 기준에 의해 일관성있게 체계적으로 적용되어야 한다. 이것을 '논리체계화'라고 한다. 프로그램 코드를 논리적으로 체계화된 형태로 만들어주기위해서는 논리단순성(logical simplicity), 논리일관성(logical consistency) 등을 도모해주어야 한다.

논리단순성은 「단순화된 것이 체계화되기 쉽다.」는 가장 기본적인 원리를 바탕으로 추구되어야 하며, 논리일관성은 「표현기준이 일관성있도록 되어있는 것이 이해하기 쉽다.」는 원리를 바탕으로 추구되어야 한다.

필요에 따라서는 De Morgan의 법칙이라든가 분배의 법칙 등의 부울대수와 관련된 연산을 수행하여 가장 논리적으로 체계화된 형태로 프로그램이 구성될 수 있도록 해주는 것이 중요하며, 역공학 자동화 도구에서는 이러한 논리체계기능의 부여가 당연히 고려되어야 한다.

3.4 시공간지역성(temporal & spatial locality)

P. Denning의 Working Set theory에 의하면 지역성(locality)에는 시간지역성(temporal locality)과 공간지역성(spatial locality)이 있는데, 시간지역성(temporal locality)이란 먼저 참조되어진 어떤 기억장소의 부분이 이후에도 계속 참조될 가능성이 높은 성질을 뜻하며, 공간지역성(spatial locality)이란 어느 기억장소가 참조되면 그 부근의 장소가 이후에도 계속 참조될 가능성이 높은 성질을 뜻한다.

이러한 성질은 데이터(data)와 프로세스(process)의 양쪽에 모두 적용될 수 있으며 재공학 도구에 이러한 시공학지역성을 감안한 함수라든가 변수등의 정리기능을 부여하는 것은 대단히 중요한 의미가 있다[33].

3.5 경계인식성(boundary recognition)

인간은 경계에 대한 인식능력을 가지고 있다. 이러한 성질을 '경계인식성'이라 한다.

소프트웨어 유지보수에 있어서도 프로그램 코드의 각 제어구조 별 및 코드의 기능별로 구분된 블럭간에 이러한 경계인식을 통한 문제영역에 대한 경계선 확정분할 조정작업 등이 필요하게 된다. 경계의 확정 및 재조정에는 여러 가지 방법이 있을 수 있겠지만, 경계의 인식에는 추상화사다리(ladder of abstraction)의 개념적용에 의해 먼저 넓은 범위를 본 다음에 지역적으로 보는 것이 무엇보다도 중요하다.

이러한 경계인식 기능을 유지보수 자동화 도구에 포함시킨다면 기존 코드에 대한 인식도 및 이해도를 대폭 향상시켜줄 수 있다.

4. SMA 구현의 핵심원리

4.1 재공학을 지원하는 문제해결 원리

재공학 기술을 적용하여 소프트웨어 유지보수를 자동화하기 위해서는 재공학기술 적용 첫단계에서 프로그램 코드의 각 부분이 문제 영역의 종류에 맞도록 하는 분류작업이 선행되어야 한다.

재공학 기술 중에서도 특히 역공학 기술을 적용하는 것이 아주 중요한 데, 역공학을 시도하기 전에 코드에 대한 재구조화 작업이 선행되어야 한다.

코드에 대한 재구조화 작업을 종래에는 순차, 선택, 반복의 세가지 개념을 가지고 GOTO문을

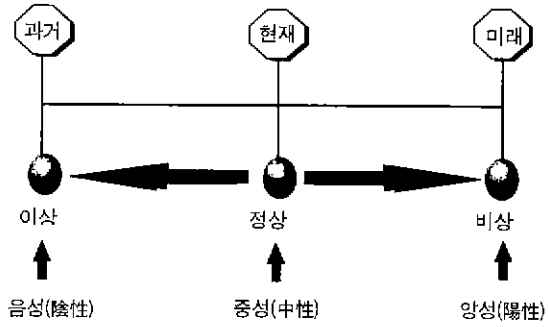
포함한 스파게티 코드(spaghetti code)를 GOTO문을 제거한 구조화된 코드로 변환하는 작업이 주로 이루어져 왔다. IBM사의 COBOL/SF(structuring Facility) 등이 바로 그것이다 [20].

코드 재구조화에 있어서 자동화비율을 높히기 위해서는 코드를 문제영역별로 정리하여 재구조화해주는 작업이 필요하다. 결국 코드를 재구조화해서 역공학으로 재문서화(redocumentation)나 설계복원(design recovery)을 행하기 위해서는 반드시 코드를 문제영역별로 분해하여 문제해결목적에 맞게 재조립해주는 작업이 필요하다.

이 때, 문제의 영역에 대한 분류는 환경의 변화를 감안한 것이 되어야 하며, 부정적인 상황에 대처하는 코드에 의한 문제영역과 긍정적인 상황에 대처하는 코드에 의한 문제영역에 대한 구분이 자동적으로 이루어질 수 있도록 하는 양면적 사고의 적용이 아주 중요하다.

문제를 세력의 균형관계를 고려하여 정상과 비정상의 양면적인 입장에서 볼 때의 영역의 구분을 나타내면 <표 7>와 같다 [1~8, 12, 13, 21, 31, 33].

문제를 계통별로 인과법칙에 따른 시계열적인 흐름의 비교를 나타내면 (그림 5)와 같다.



(그림 5) 문제의 성질별 시계열적 흐름의 비교

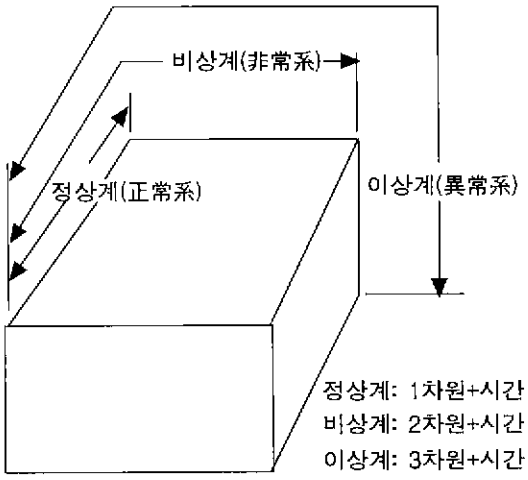
세력의 균형관계적인 측면에서 3차원적인 공간의 표현과 시계열적인 전진과 정지와 퇴보를 표현해 주기 위해서는 종래의 나무구조(tree structure)와 같은 2차원 구조만으로는 표현이 불가능하기 때문에, 잔디구조(lawn structure)와 같은 새로운 형태의 4차원 구조의 적용이 필수적이다.

<표 7> 세력균형관계를 감안한 문제영역의 구분

문제영역의 구분		내 용
정상적문제 (正常的問題)	정상계 (正常系)	- 시스템이 목적을 세워 해결하는 문제 - Dijksra의 이론이 적용되는 계(system)로서 1차원적인 선의 흐름에 해당한다. - 정상적으로 목적을 세워서 해결해나가는 계이다. - 계열 및 계층간의 건너뛰기는 1단씩만 허용된다. - 시간적으로는 정지로 본다.
비정상적문제 (非正常的問題)	비상계 (非常系)	- 시스템 스스로 해결할 수 있는 문제 - H이론이 적용되는 계로서 2차원적인 평면에 해당한다. - 비상시와 같은 급격한 환경의 변화에 적용되는 계이다. - 반드시 2단이상의 계열간 건너뛰기가 존재한다. - 계열간의 건너뛰기는, 2차원 평면상의 작은 계열에서 큰 계열로만 가능하다. - 시간적으로는 빠른 시간의 흐름으로 본다.
	이상계 (異常系)	- 시스템 스스로 해결할 수 없는 문제 - H이론이 적용되는 계로서 3차원적인 공간에 해당한다. - 시스템 자체에 스스로 해결할 수 없는 장애가 발생한 경우에 해당한다. - 비상계와 같이 2차원 계열간 뿐만이 아니라 3차원적 계층간의 건너뛰기가 이루어진다. - 시간적으로는 시계열적인 후퇴(시간을 거슬러 올라감)가 동반된다.

잔디구조라 함은 인간의 공간적인 사고의 습성에 부응하는 쪽으로 만들어진 문제해결을 위한 인공환경으로, 시각적으로는 3차원적인 모습을 하고 있으며, 개념적으로는 시간의 전진, 정지, 퇴보의 개념이 추가되어 전체적으로 4차원적인 환경구조를 가진 문제해결 구조이다.

문제의 해결구조인 (그림 4)와 같은 잔디구조(lawn structure)를 각 계(系)에서 문제에 대처하는 차원의 관점에서 나타내면 (그림 6)과 같다.



(그림 6) 문제대처 차원의 관점에서의 잔디구조

잔디구조(lawn structure)라 함은 3차원적인 공간요소에다가 시간요소를 첨가하여 4차원적으로 문제를 풀어나갈 수 있도록 인간두뇌의 문제해결구조를 모형화한 시계열적인 흐름을 가진 인공 환경이다.

여기서 '시계열적 흐름'이란 '정상적인 시계열 흐름을 기준으로 하는 상대적인 시계열 흐름의 속도' 를 뜻하는 것이다. 예를 들어, 정상적인 나이가 40살이라 하더라도 교통사고로 인해 이상이 발생하여 10살 정도의 행동밖에 할 수 없다고 한다면 시계열적으로 30년의 후퇴로 본다.

소프트웨어 유지보수 작업을 자동화함에 있어서는 프로그램 코드를 잔디구조라는 개념적인 인공 환경구조상에서 분해 배치하여, 코드의 성질을 요소별로 정확하게 분류하는 공정이 자동화 프로그램 속에 들어감으로써, 대형 프로그램이라 할지라도 손쉽게 복잡도를 제어하여 유지보수할 수 있게 된다. [9~19, 23~33].

4.2 재공학을 지원하는 요소의 절단·조합 원리

소프트웨어 유지보수를 자동화하기 위해 요소별 절단(slicing)과 조합을 성공적으로 행하기 위해서는 각 요소에 대한 성질별 구분작업이 필요하다.

일단 요소(component) 별로 절단(slicing) 작업이 자동화되면, 요소에 대한 수정 작업 또한 수월해지게 된다. 따라서, 기존 요소부품에 대한 재사용(reuse)도 무리없이 진행될 수 있게 된다.

재사용(reuse)은 각 부품별로 기본부품으로부터 기능부품은 물론 모듈부품에 이르기까지 필요한 추상화의 정도에 따라서 유지보수 작업이 진행되는 동안에 적절히 행해질 수 있다.

유지보수 작업을 자동화 할 때 소프트웨어 부품을 재사용 작업을 진행해줌에 있어서 미리 숙지해두면 좋은 기본 용어들을 정리하면 <표 8>과 같다.

소프트웨어의 재사용에 필수 불가결하게 요구되는 기술이 바로 '유지보수(maintenance)' 기술이다.

Maryland University의 V.R. Basili는 재사용부품을 만들기 위한 유지보수모델(maintenance model)을 <표 9>와 같이 분류하였다.

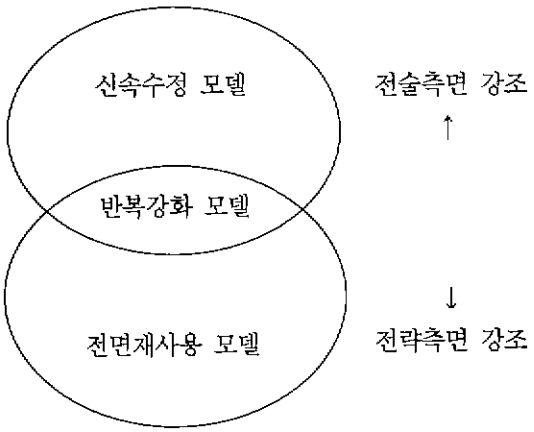
3가지 모델을 전략과 전술의 관점에서 비교해보면 (그림 7)과 같다 [20].

<표 8> 소프트웨어 부품관련 용어의 정의

용 어	내 역
S/W 부품 가공 (component customize)	- 보관되어있던 S/W부품을 새로운 S/W환경에 맞도록 약간 수정하는 것.
S/W 패턴(pattern)	- 프로그램 함수 등에서 특정 변수이름, 특정 계산식 등만 직접 사용자가 삽입하면 즉시 재사용이 가능하도록 준비해 놓은 것. - Texas Instruments사의 CASE도구인 IEF(Information Engineering Facility)에서는 템플릿(templates)이라고 부르기도 한다.
S/W 뼈대(skeleton)	- S/W의 골격 - 뼈대만 완성해 놓은 틀 프로그램 - 재사용되기 위해서는 부품가공(customize)이 필요하다
검은상자 부품 (black box component)	- 내부의 구조를 들여다 보거나 수정을 가할 수 없도록 만들어진 S/W 부품 - 목적코드(object code)로 만들어진 라이브러리 등이 이에 해당한다. - 사용자에 대해 정보은폐(information hiding)가 되어있다. - 명확하게 정의된 인터페이스를 통해야만 접근이 가능하다.
흰상자 부품 (white box component)	- 내부의 구조를 보고 수정을 가할 수 있도록 만들어진 S/W 부품 - 패턴(pattern)이라든가 템플릿(template)은 흰상자 부품에 해당한다. - 무엇(what)을 , 어떻게(how) 하는 것인지를 명확하게 알 수 있다.
S/W 빌딩 블록 (building block)	- 하드웨어 칩(chip)과 유사한 성격을 가진 S/W부품 - 하드웨어 칩처럼 인터페이스가 명확하게 정의되어 있다. - 사양부와 실현부로 구성되며, 사양부에는 기능 및 사용자 인터페이스가 기술되고, 실현부에는 실현코드(implementation code)가 기술된다. - 독일 IBM사의 Boeblingen연구소에서 이와 관련한 도구로서 BB/LX(Building Block/ Language eXtension)을 개발한 바 있다.

<표 9> 재사용부품 제작용 유지보수 모델

모델의 구분	내 역
신속수정 모델 (quick-fix model)	- 필요성이 생길 때마다, 즉시 원시코드(source code)를 검토하여 수정한 뒤 바로 컴파일하여 목적코드(object code)를 생성시키고는 관련된 사양서 등의 수정작업을 행하는 모델.
반복강화 모델 (iterative enhancement model)	- 재사용부품을 만들 때 영향을 받을 수 있는 설계부분 등 상위의 사양서 등의 수정작업을 먼저 행한 뒤, 그 변경을 하위의 코드로까지 연결시키는 모델
전면재사용 모델 (full-reuse model)	- 새로운 시스템을 개발함에 있어서의 문제해결을 이전에 개발했던 유사한 시스템의 개발과정에서 얻어진 해결책의 집합의 재사용을 통해 도모하는 모델



(그림 7) 유지보수 모델의 구분

신속수정 모델(quick-fix model)은 너무 임기응변식의 작업이 강조됨으로써, 유지보수작업을 통해 재사용부품을 만듦에 있어서, 문서화 등 설계쪽의 상위사양에 소홀하기 쉬워, 일단 만들어진 재사용부품에 대해서 그 품질을 신뢰하기 어려운 단점을 가지고 있다.

또한, 전면재사용 모델(full-reuse model)의 경우는 이상적이기는 하지만 너무 장기적 개발을 염두에 둔 전략적 측면이 강하여 쉴새없이 변화하는 최근의 소프트웨어 개발환경에 쉽게 적응하기 어려운 문제점을 내포하고 있다.

따라서, 현시점에서는 어느 정도의 전략적인 측면과 전술적인 측면을 적절히 고려하고 있는 반복강화모델이 상당히 긍정적인 면을 가지고 있다고 볼 수 있다.

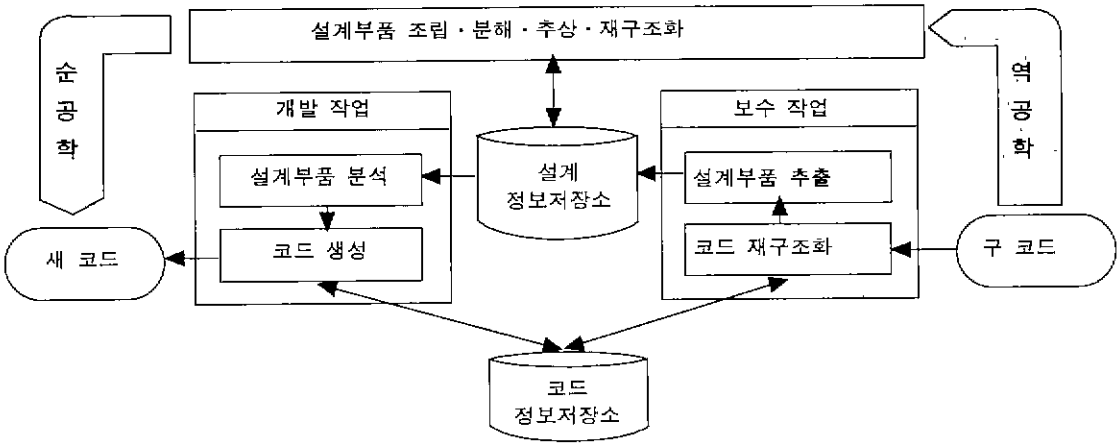
이러한 반복강화모델은 시스템을 진화적인 측면에서의 재설계를 용이하게 하고, 보다 신뢰성있고 품질좋은 재사용부품으로 만들어서 장래에 재사용하기 위해 끊임없는 설계사양등에의 유지보수작업을 통해 뼈대(frame)를 강화해 나갈 수 있는 장점도 가지고 있다.

유지보수를 자동화 하는 소프트웨어 재공학 도구의 구성은 (그림 8)과 같다.

유지보수 대상이 되는 코드를 역공학 자동화로 재문서화내지는 설계복원을 행하기 위해서는 기본 코드의 각 요소를 원자수준에서 절단(slicing)해 주는 작업이 중요하다.

코드는 절단되어 <표 10>과 같은 기본부품으로 정리되어 정보저장소(repository)에 저장된다.

기본부품은 역공학 추상화 작업을 위해 기본부품의 용도별로 묶여서 덩이부품(block component)으로 조립된다. 기본부품이 조합되면 <표 11>과 같은 덩이부품이 된다.



(그림 8) 재공학 도구의 구성

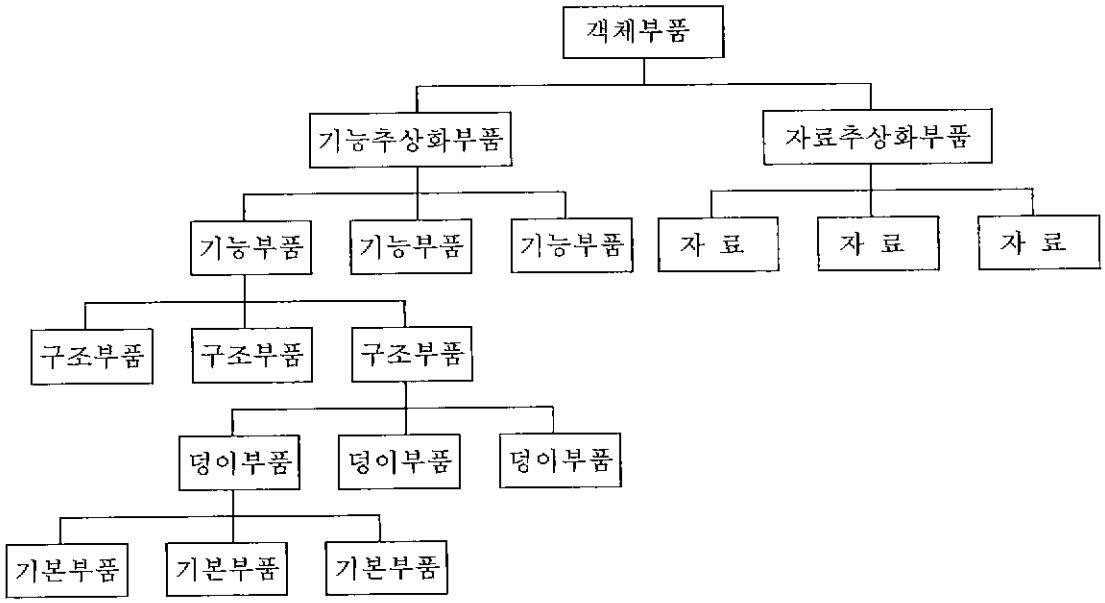
<표 10> 기본부품의 구분

기본부품의 구분	내 역
목적형 기본부품	제어구조별 목적부분을 형성하는 부품
기능형 기본부품	입·출력, 처리, 하청, 주석 등을 형성하는 부품
보조형 기본부품	제어구조에 종속된 보조기능을 형성하는 부품
제어형 기본부품	문서화 작업에 필요한 제어작업 수행용 부품
흐름형 기본부품	제어흐름의 구분을 나타내는 부품

덩이부품이 조립되면 이 덩이부품을 이용하여 원하는 구조부품을 만들어낼 수가 있다. 역공학 자동화 도구는 프로그램 코드를 절단(slicing)하여 기본부품(basic component)을 추출해내고 이들을 취합하여 덩이부품(block component)를 만들어내며, 덩이부품을 구조부품(structure component)으로 구조부품을 기능부품(function component)으로,기능부품을 보다 상위인 객체부품(object component)으로 (그림 9)와 같이 추상화시키는 과정을 통해 순식간에 재문서화(redocumentation)내지는 설계복원(design recovery)의 목적을 달성할 수 있게된다 [31, 32, 33].

<표 11> 덩이부품의 구분

덩이부품의 구분	내 역
바탕 덩이부품	분석되는 모듈이 주모듈(main module)인가 부모듈(sub module) 인가를 나타내어 주는 건축물의 기초공사에 해당하는 설계 바탕을 형성해주는데 사용되는 부품
목적 덩이부품	프로그램의 각 제어구조의 종류와 함께 구조의 목적을 우측에 기술할 수 있도록 해주는 부품
조건 덩이부품	프로그램의 제어구조가 어떤 조건에 따라 처리를 행하는 if~then 이라든가 for~do 등과 같은 조건을 가진 구조일 경우에 이와 같은 조건을 표시해주는 부품
처리앞 덩이부품	프로그램 코드의 각 구조의 종류에 따라 구조의 처리부분이 시작됨을 알려주기 위한 도입부를 구성하는 부품
처리속 덩이부품	프로그램 코드의 각 구조의 구체적인 처리를 기술할 수 있도록 하는 부품
처리끝 덩이부품	프로그램 코드의 각 구조의 처리부분이 마감되어 흐름이 다른 구조로 넘어감을 나타내어 주기 위한 각 구조의 종결부를 표현해 주는 부품
보수 덩이부품	건물을 조립식으로 건축하거나 분해할 때 보수하기 위한 자재가 필요하듯이 프로그램 코드의 각 구조 또는 블럭 등이 조립되거나 추상화되거나 분해되는 과정에서 각 구조의 종류에 따라 허술해 지는 부분을 보수해주기 위한 부품



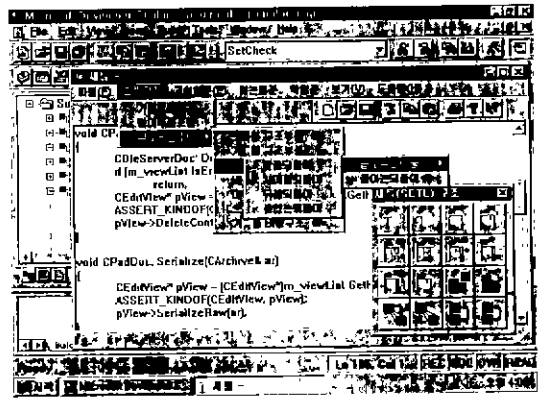
(그림 9) S/W부품조립의 계층도

5. SMA 성공사례와 향후 과제

재공학의 전공정을 지원하여 유지보수를 자동화시키기 위해서 순수한 국내기술로 탄생한 자동화 도구가 '구조화능률도구(構造化能率道具)'라는 공식 명칭을 가지고 있는 새틀(SETL: Structured Efficiency TooL)이다. 새틀은 우리말로 '새로운 틀' 즉 신도구(新道具, new tool)라는 뜻의 발음이 된다. 새틀은 (그림 8)과 같은 재공학 도구의 전공정을 지원하도록 개발된 자동화 도구이다.

(그림 10)은 새틀 윈도95판의 모습을 나타낸 것이다.

이제까지의 다른 도구와는 달리 100% 순수 국내기술만으로 개발된 새틀(SETL)은 순공학(forward engineering), 역공학(reverse engineering) 및 재구조화(restructuring)를 포함한 재공학(reengineering)의 전공정에 대한 완벽한 지원을 성공적으로 수행하고 있다.



(그림 10) 새틀 윈도95판이 Visual C++과 연동되는 모습

특히 무수한 GOTO문을 포함한 스파게티 코드도 순식간에 구조화시키는 강력한 재구조화 능력이 실험을 통해 입증되고 있다(정보처리학회지 96.1월호, 사례발표).

향후 새틀(SETL)과 같은 재공학도구(reengineering

tool)가 일반화되기 위해서는 다음과 같은 과제의 해결이 필요하다.

첫째, 다양한 언어의 지원이다. 현재 새틀이 지원하는 언어는 C/C++과 BASIC언어이다. 향후 빠른 시일내에 Object Pascal(DELPHI), JAVA등과 같은 언어는 물론 현재 공공기관에서 많이 사용되고 있는 Cobol 등과 같은 언어까지 지원하게 되면 유지보수의 자동화에 일대 전기가 마련될 수 있을 것이다.

둘째, GUI기능의 강화이다. 재공학의 전단계를 마치 레고브럭을 가지고 조립분해하는 것처럼 가상적인 인지환경을 시각적으로 조성하는 기능이 강화됨으로써 방법론의 적용과 도구의 사용이 보다 쉬워지게 될 것이다.

셋째, 요구분석 자동화도구와의 결합이다. 새틀에 요구분석을 자동화하기 위한 기능이 추가되어 요구분석↔설계↔구현의 전공정이 연동되도록 함으로써 개발과 유지보수의 자동화를 완결시키게 되면, 소프트웨어의 유지보수 자동화를 중심으로 하는 재공학 기술의 일반화는 더욱 촉진될 것이다.

참고문헌

[1] CORRADO BÔHM AND GIUSEPPE JACOPINI, "Flow Diagrams, Turing Machines And Languages With Only Two Formation Rules", Communications of the ACM, Volume 9/ Number 5/ May, 1966, page 366~371

[2] EDSGER W. DIJKSTRA, "Letters to the Editor(Go To Statement Considered Harmful)", Communications of the ACM, Volume 11/ Number 3/ March, 1968, page 147~148

[3] DONALD E. KNUTH, "Structured Programming with go to Statements", Computing Surveys, Vol. 6, No. 4, December 1974, page 261~301

[4] Bruan W. Kernighan and P.J. Plauger, 木村 泉 譯, "プログラム書法(The Elements of Programming Style)", 共立出版株式會社, 1985

[5] 花田收悅, "プログラム設計圖法", 企劃センター, 1989

[6] ROBERT J. RADER, 池浦孝雄 外 3人 共譯, "ソフトウェア設計(ADVANCED SOFTWARE DESIGN TECHNIQUES)", 共立出版株式會社, 1983

[7] Edsger W. Dijkstra and W.H.J. Feijen, 玉井 浩 譯, "プログラミングの方法(A METHOD OF PROGRAMMING)", サイエンス社, 1991

[8] M.A. JACKSON, "構造的プログラム設計の原理(PRINCIPLES OF PROGRAM DESIGN)", 日本コンピュータ協會, 1980

[9] James Martin & Carma McClure, 國友義久・渡邊純一 譯, "ダイアグラム法による ソフトウェア構造化技法 (Diagramming Techniques for Analysts and Programmers)", 近代科學社, 1991

[10] 廣松恒彦, "プログラムの設計・作成", オーム社, 1981

[11] 國友義久, "プログラム設計徹底マスター", SOFTBANK BOOKS, 1991

[12] 西坂秀博, "計劃はどのように作るか", 共立出版株式會社, 1991

[13] 佐藤允一, "問題構造學入門", ダイヤモンド社, 1989

[14] 大塚純一, "仕事を圖解する技術 フロ-チャ-ト入門", 日本能率協會, 1990

[15] 上野一郎, "能率の考え方", 産能大學, 1990

[16] 森谷宜暉・山下福夫, "問題解決型 業務改善の考え方・進め方", 産能大學出版部, 1990

[17] 鯉沼 章, "フロ-チャ-トによる事務分析", 日刊工業新聞社, 1991

[18] 馬場 勇, "ソフトウェア開發實踐技法", 技術

評論社, 1989

[19] James Martin, 松山一郎 譯, "自動システム設計のための標準ダイアグラム作成技法(RECOMMENDED DIAGRAMMING STANDARDS for ANALYSTS and PROGRAMMERS - A BASIS for AUTOMATION", 近代科學社, 1991

[20] 竹下 亨, "ソフトウェアの保守・再開發と再利用", 共立出版株式會社, 1992

[21] 下村隆夫, "プログラム スライシング技術と應用", 共立出版株式會社, 1995

[22] CARMA McCLURE, "THE THREE Rs OF SOFTWARE AUTOMATION", Prentice Hall, 1992

[23] 加藤光明 外, "The BUG", Ohmsha, 1995

[24] 竹內元一 外, "圖解する技術", 中經出版, 1994

[25] Washida Koyata, "思考の技術發想のヒント", 日本實業出版社, 1996

[26] 花田收悅, "ソフトウェアの仕樣化と設計", 日科技連, 1992

[27] 龜崎ヤスナオ, "創造性の開發", 産能大學, 1996

[28] 森谷 ヨシテル, "現代と能率", 産能大學, 1996

[29] 原田賢一, "構造エディタ", 共立出版株式會社, 1988

[30] 유홍준, "한국형 순서도 쎄크", 도서출판 홍은, 1992

[31] 유홍준, "인간지향 무른모 설계 방법론", 도서출판 홍은, 1997

[32] 유홍준, "S/W 공장 자동화의 구현 원리", 마이크로 소프트웨어, 1995.11～1996.1

[33] 유홍준, "S/W 유지보수의 실전 기법", 마이크로 소프트웨어, 1996.2～1996.9.

[34] 日本情報處理學會, "情報處理 ハンドブック, 1989

유 홍 준

1992년 일본 산능단대 능률학과 졸업(정보처리 전공)
1983년 1급통신사(무선통신 분야)
1994년 정보관리 기술사(정보처리 분야)
1994년 기술지도사(정보처리 분야)
1977년-1981년 한국특수선(주) 통신장
1981년-1985년 범양상선(주) 통신장
1988년 제5회 전국PC경진대회 공모부문 응용소프트웨어 분야 1위(상공부장관상 수상)
1985년-현재 홍준 정보처리 학원 원장
1989년-현재 도서출판 홍은 대표
1994년-현재 총무처 정부전자계산소 전자계산교육원 외래강사
1997년-현재 힘스미디어 대표
저 서 : 객체지향 개념, 텍스트애니메이션, 마이크로 컴퓨터 음향기법, 프로그램 보호기법, 한국형 순서도 SEC의 모든 것, 무른모 설계 자동화 방법론, 그래픽 기법 등 다수
관심분야 : 소프트웨어 공학(특히, 순공학, 역공학, 재구조화, 재공학, 소프트웨어 공장자동화, 소프트웨어 프로세스 리엔지니어링)