特輯 技術解說

소프트웨어의 再使用 技術

李 京 煥
中央大學校 電子計算學科

Ⅰ. 再使用의 필요성

80년대에 들어와서 소프트웨어의 규모와 그 개발 비용이 매우 커져가고 있으며, 특히 유지보수 비용은 대단하다. 이러한 현상은 다음과 같은 몇가지 이유로 설명할 수 있다.

첫째, 하드웨어의 대량 보급과 컴퓨터 적용 분야의 확산으로 인하여 수요가 매우 증가하고 있으나 소프트웨어를 개발할 전문 인력은 매우 부족한 상태이다. 1982년에 발표된 미 국방성의 자료에 의하면 소프트웨어의 개발 요구는 해마다 12% 늘어나는데 반해 전문 인력은 연 4% 정도밖에 증가하지 않으므로 소프트웨어의 수급 불균형은 시간이 갈수록 커지며 이로 인하여 그 개발비용이 확대되고 있다.

둘째, 컴퓨터의 응용분야가 확대 되어감에 따라 개발되어지는 소프트웨어의 규모와 수준이 이전보다 훨씬 복잡하고 어려운 내용의 것이 많아서 응용분야를 이해하여 복잡한 소프트웨어를 개발하는데 많은 시간과 인력이 요구된다.

셋째, 소프트웨어 개발을 부드럽게 이끌어 주는 툴과 방법론이 부족하다.

넷째, 소프트웨어 유지 보수량이 늘어나서 이에 대해서 많은 인력과 시간이 소요되고 있다.

이상과 같은 소프트웨어의 특성때문에 소프트웨어의 비용중에서 개발비용이 30%인 반면에 유지보수 비용은 70%까지 증가하고 있다. E. B. Swanson의 보고에 의하면 소프트웨어의 유지보수를 해야되는 요인을 보면 표 1과 같다.

표 1의 통계에서 사용자의 요구가 확대되어 유지보수를 하는 비중이 41.0%인데, 사용자 요구 변경은 다시 세분해보면 표 2와 같다. 새로운 레포트의 추가가 40.8%이므로 이것을 유지보수의 비율에서 보면 16.7%이고, 소프트웨어의 비용 전체에서 보면 11.7%가 되므로, 만일 再使用의 技術에 의해서 사용자의 요구 확대를 처리해 줄 수 있다면 전체 소프트웨어의 비용중에서 11.7%의 생산성을 향상시킬 수 있을 것이다.

소프트웨어의 유지보수의 경제성 및 효율성을 도모하기 위해서는 기존의 시스템을 보수하기 위한 전략, 앞으로의 시스템을 개발하기 위한 전략, 유지보수 관리자등을 지도하기 위한 전략, 소프트웨어의 효율적 관리를 위한 전략이라는 차원에서 계획성 있게 다루어져야 한다.

표 1. 유지보수의 원인별 비중

보 수 원 인	비중(%)
사용자를 위한 확장	41.0
입력 데이타 파일의 변경	17.4
긴급한 프로그램의 수정	12.4
Routine적인 Debugging	9.3
하드웨어 및 소프트웨어의 변경	6.2
Document 수정	5.5
처리 효율 향상을 위한 재코딩	4.0
기 타	4.2

표 2. 사용자의 요구 확장에 따른 유지보수의 비율

사용자의 요구 변경사항	비율(%)
새로운 레포트 추가	40.8
기존 레포트에 데이타 변경	27.1
기존 레포트의 Layout 변경	10.0
기존 레포트의 수를 감소시키기 위한 압축	6.4
기존 레포트의 데이타 압축	5.6

100% 신뢰성 있는 시스템을 만든다는 것이 현실적으로 불가능한 까닭에 유지보수가 용이하도록 소프트웨어를 개발해야 한다는 점. 즉 생산성 향상과 함께 품질 관리하는 측면에서 바람직한 방향으로 유지보수 비율을 줄일 수 있는 방향이 모색되어야 한다.

이와 같은 유지보수의 문제를 해결할 수 있는 기술을 다음과 같이 정리할 수 있다.

- 구조기술의 이용
- 자동화된 tool 사용
- 4 세대 언어 및 generator 사용
- 데이타 베이스 기술 사용
- 경험 축적
- 현대화된 소프트웨어 技術

현대화된 소프트웨어 技術중에는 소프트웨어의 再使用(reuse)기술을 많이 채용하고 있다.

Ⅱ. 再使用의 技術과 그 對象

소프트웨어의 개발 비용과 그 급증하는 문제를 해결하기 위해 활발히 연구하고 있는 분야가 소프트웨어 재사용을 통한 문제해결 방법이다. 우리가 일상적으로 작성하고 있는 소프트웨어를 살펴보면 반복적인 부분이 매우 많음을 알 수 있다. 즉 새로운 소프트웨어를 작성할 때 이전에 작성했던 부분을 다시 작성하게 되는 경우가 많은 것이다. 예를 들면, 정렬, 탐색 등의 부분은 많은 소프트웨어에 중복되어 나타나고 있다. 이미 개발되어 있는 소프트웨어와 새로 개발할 소프트웨어가 특정 부분에 있어서만 독특하고 나머지는 대체로 같거나 비슷하다는 점이 바로 소프트웨어의 재사용 방법론이 매우 유망하다는 사실을 인식시켜 준다. 소프트웨어를 재사용하고자 할 때 그 재사용 대상은 다음과 같은 것들이 될 수 있다.

(1) 코드조각(code fragments)

프로그래머의 가장 일차적인 산물인 코드 (프로그램)의 재사용은 가장 오래된 소프트웨어의 개발 기술로 발전되어 왔다.

(2) 논리적 구조(logical structure)

프로그램의 처리 순서와 자료 구조로 구성된 구조로서 부품(모듈, 자료의 모임)들과 그들 사이의 관계(calling, parameter passing, inclusion)를 정의하는 것으로 시스템의 내부 설계 과정에서 개발된다. 일반적으로 재사용은 대응된 자료를 변화시키면서, 논리적 구조(프로그램과 자료 구조)를 재사용하게 된다.

(3) 기능적 구조(functional architectures)

내부 설계에 익숙하지 않은 사용자를 위해서 그 모듈(부품)의 기능을 설명한 외부 설계로서 기능(function)과 데이타 객체(data object)들의 명세를 규정하여 재사용한다. 기능적 구조의 재사용은 기능적 집합(functional collection)과 공통시스템(generic system)을 들 수 있다.

- 기능적 집합 : 응용분야 함수들의 집합으로 수학적 서브루틴의 패케지를 예로 들 수 있다.
- 공통시스템 : 한 응용분야의 함수 집합에 많은 부가적인 함수들을 통합하여 대응된 값에 따라서 여러가지 기능을 갖도록 하는 핵심 시스템을 말한다.

(4) 외부영역 지식(external knowledge)

외부 영역 지식은 소프트웨어 개발의 대상 영역과 소프트웨어를 개발하는 기술에 관한 지식을 말한다.

- 응용 분야 지식(application area knowledge) : 컴퓨터를 사용하는 실질적인 영역의 지식으로 회계관리나 호텔 경영의 지식 등을 예로 들 수 있다.
- 개발지식(development knowledge) : 소프트웨어를 개발하는 절차에 관한 정보로 생명주기 모형(life cycle model), 제품의 정의(product definition), 검사계획(test plan), 품질보증 목록표(quality assurance checklist) 등을 예로 들 수 있다.

(5) 재사용 데이타(reusable data)

응용분야나 시스템에서 데이타를 레코드나 화일로 통합하여 재사용하는데 이때 고려하여야 할 점은 다음과 같다.

- 한 시스템에서 다른 시스템으로 데이타를 쉽게 운송할 수 있기 위해 표준적인 상호 교환 형식이 요구된다.
- 재사용을 돕기 위해 데이타 생성, 삭제, 수정등 변화 추이(자료 변천의 역사)의 기록이 필요하다.
- 재사용율이 높은 데이타일수록 안정성을 더 많이 고려하여야 한다.
- 많은 양의 데이타를 저장하기 위해 값싸고 충분한 저장 용량을 갖는 저장 매체의 고안이 필요하다.

(6) 설계의 재사용(reusable design)

소프트웨어를 구성하는 소프트웨어 아키텍쳐-모듈들의 상하 구조, 데이타의 구조에 관한 정보의 재사용으로 소퓨트웨어의 영역별 기능, 설계 단위별로 대응시킬 수 있는 객체에 관한 정보가 필요하다. 이를

위해서는 설계 툴을 표준화하고 영역을 잘 분류하여 분석해야 된다.

(7) 프로그램의 재사용(reusable program)

소프트웨어 시장에 현존하는 프로그램을 효율적으로 재사용하는 데에는 소프트웨어들 사이에 작동(프로그램 기능의 시작과 끝마침, 콘트롤 등)의 사용자 인터페이스(user interface)의 표준화가 필요하다.

(8) 공동시스템(common system)

재무분야, 인간자원, 판매 보고서 작성 영역은 60%에서 70%까지 공동 시스템의 지원이 가능하지만 제조 분야나 공학 분야는 아직 미흡하다. 그러나, 재사용 기술의 발전으로 제조 공정의 자동화 개념을 소프트웨어 개발에 도입할 수 있게 될 것이다.

(9) 모듈의 재사용(reusable module)

프로그램의 패턴화나 변환에 의해서 표준화된 함수나 서브루틴이 재사용은 되고 있지만, 좀 더 효율적인 재사용을 위해서는 라이브러리 관리 시스템의 지원이 있어야 한다.

Ⅲ. 객체 指向 技術

再使用의 技術은 구조화 프로그래밍, CASE, prototyping 技術 등과 함께 결합하여 발전하여 왔다. 그 중에서도,

- 일반적인 함수의 再使用
- 파일 전송에 의한 再使用
- 일반적인 구성요소에 의한 再使用
- 프로그램 변환에 의한 再使用

등이 사용되어 오다가, Ada언어가 개발되면서 부터 抽象資料型(ADT : abstract data type), 정보은익(information hiding) 및 package 등의 기술을 적용할 수 있게 되었다.

특히 Smalltalk과 C++ 언어가 소프트웨어 개발에 적용되면서부터 encapsulation 및 inheritance 기법을 적용함으로써 再使用의 효율을 높히기 시작하였다.

1) 객체(object)

객체의 개념은 1960년대 개발한 Simular 언어에서 도입되었다. Simular는 모듈화에 매우 큰 관심을 가지고, 모듈을 다른 프로그램처럼 프로시듀어에 기초하지 않고, 실질적인 객체(physical object)를 기반으로 작성할 수 있도록 하였다. 객체별로 갖고 있는 (분류할 수 있는) (behavior) 범위를 정해서 모듈로 작성하므로 한 객체는 관련된 프로시듀어와 데이타를 모아둔 페케이지가 된다. 이 프로시듀어를 메쏘드(method)라 하고 자료값들이 대응되는 것을 변수(variable)라고 말한다.

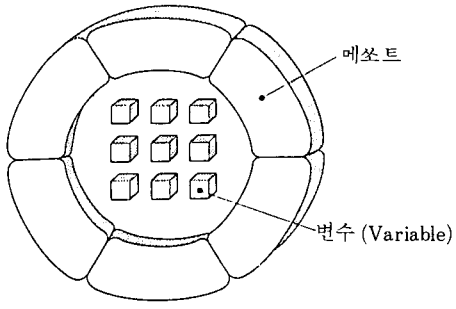


그림 1. 객체

예 : 크루스 콘트롤을 가진 자동차

자동차를 한 객체로 표현하기 위해서는 자동차의 행동(움직임, 짐 싣고 내림 등)을 메쏘드로 하고, 그 특성(톱니바퀴의 크기, 최고속도 등)을 변수로 하여 정의한다.

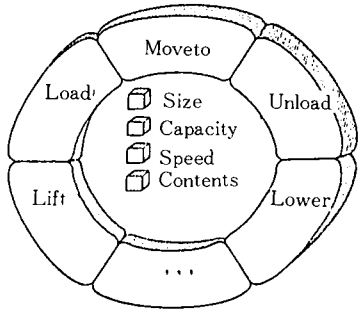


그림 2. 크루스 콘트롤의 객체

2) 메시지(message)

실제의 객체는 다음과 같은 여러가지 원인에 의해서 수 많은 변화의 영향을 받는다.

- 생성, 파괴, lifting, attaching, buying, bending, sending 등.

이와 같은 변화를 주는 것은 Simular에서는 메시지라고 표현했다. 메시지에 의해서 그 객체가 수행할 작업을 인지하게 되고, 메시지는 파라미터(para-

meter)라고 불리우는 자료요소의 모임을 가지고 메쏘드에 정보를 제공한다. 메시지를 만든 객체를 sender라 하고, 메시지를 받는 객체를 receiver라고 한다.

예 : 자동차를 움직이기 위한 메시지 –

vehicle 104 moveTo : binB7

vehicle 104는 receiver이고, moveTo는 수행할 메쏘드이며, binB7은 receiver가 가야될 장소를 말해주는 파라미터이다.

그림 vehicle에 보내는 메시지 : vehicle104 moveTo : binB7

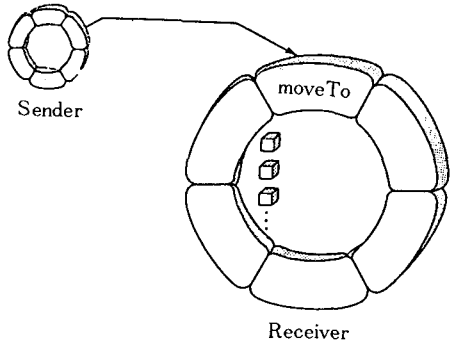


그림 3. Vehicle에 보내는 메세지

3) 클래스(class)

클래스는 메쏘드와 변수를 정의하여 객체의 구체적인 타입으로 정해진 것을 말한다. 클래스에 구체적인 값을 대입하여 특정한 클래스를 만드는데, 대응된 값을 포함한 클래스를 그 클래스의 인스턴스(instance)라고 말한다.

Automated vehicle은 하나의 클래스이고, 여기에 여러가지 값들을 대응시켜서 vehicle101, vhicle102, … 라고 부른다.

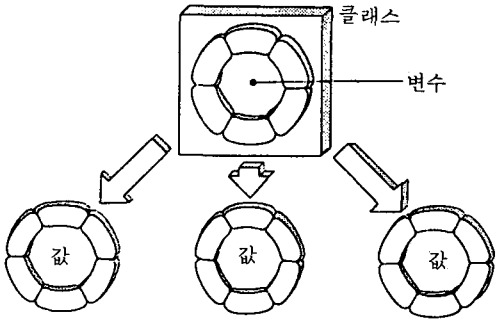
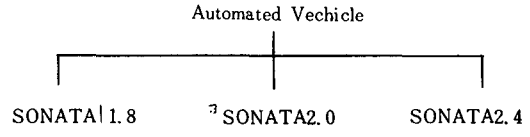


그림 4. 클래스와 그 인스턴스

4) 승계(inheritance)

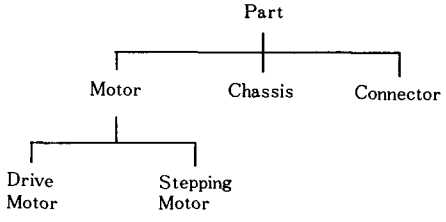
객체들 중에 한 클래스가 보다 일반적인 클래스의 메쏘드와 변수를 상속받아서 일반적인 클래스의 속성과 기능을 발휘하는 것을 승계라고 말한다. 이때 일반적인 클래스를 superclass라 하고, 상속받은 클래스를 subclass라고 말한다.

예 : automated vehicle의 subclass들



이와 같은 승계는 계층별로 조직화시킬 수 있는데 이것을 caass hierarchy라고 말한다.

예 :



5) 객체 지향언어

객체 지향 언어는 여러가지가 있으나 그 중에서도 Smalltalk와 C++이 널리 사용되고 있다. 객체 지향언어로 프로그램을 작성하기 위해서는 먼저 객체 모델링(object modeling)을 하여 객체 카드를 external spec으로 바꾼 다음에 코딩한다. 한편 再使用 시스템이 개발되었다면 객체카드를 보고 再使用할 클래스를 찾아서 객체를 만들어 再使用하고 클래스가 없으면 external spec을 작성하고 코딩하게 된다.

Ⅳ. 객체의 모델링

객체의 모델링은, ①요구정의, ②conceptual modelling, ③external modelling, ④객체지향언어로 코딩의 단계를 거쳐서 만들어진다. 모델링의 중심은

conceptual modelling인데 객체 지향 분석과 설계의 단계를 포함한다고 볼 수 있다.

LO : 모델링의 과정

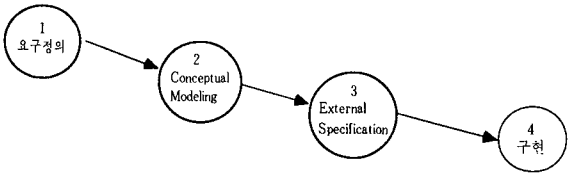


그림 5. 모델링의 과정

Conceptual modelling의 단계를 상세하게 표현하면 그림 6 과 같다. 요구정의가 끝나면 기본적인 activity script를 만들 수 있다.

L1 : 2

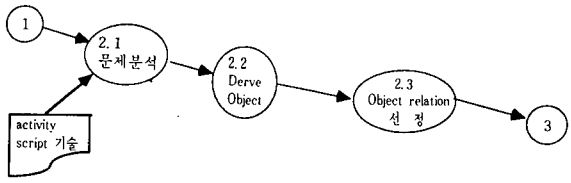


그림 6. Conceptual modelling

Activity script를 가지고 object와 그 behavior 및 attribute를 정의하면 object의 behavior와 attribute를 식별하여 설명할 수 있다.

L3 : 2.2 (drive object)

다음 단계에서는 object glossary, behavior glossary 및 feature glossary를 가지고 initiator와 participant의 behavior를 구분하여 정의하고 traceabillty를 정하여 후에 object에 관련한 정보가 빠졌거나, 중복성 및 일의성 등을 체크할 수 있도록 한다. 그 결과로서 나오는 것이 곧 modelling card이다.

끝으로 object relation을 확인하여 modelling card의 revision을 수행한다. Object relation은 그림 8 과 같이 is-a hierarchy와 class/instance 관계를 설정하는 일반화와, aggregation에 의한 클래스의 포함관계, 그리고 link와 association 관계를 정의한다.

L3 : 2.3 (object relation : modelling card revising)

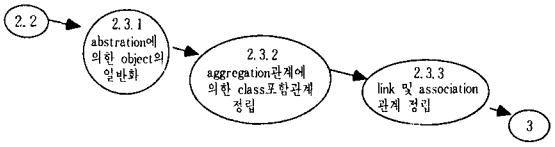


그림 8. Object relation 정의과정

Object relation 단계가 끝나면 external spec을 작성하는데, 이것은 C++와 같은 객체지향언어의 pseudo code 형식으로 표현한다.

이렇게 하여 再使用 시스템에서 再使用 클래스를 찾아서 再使用하거나, external spec을 보고 객체지향 언어로 코딩하고 새롭게 코딩된 프로그램은 再使用 라이브러리에 등록한다.

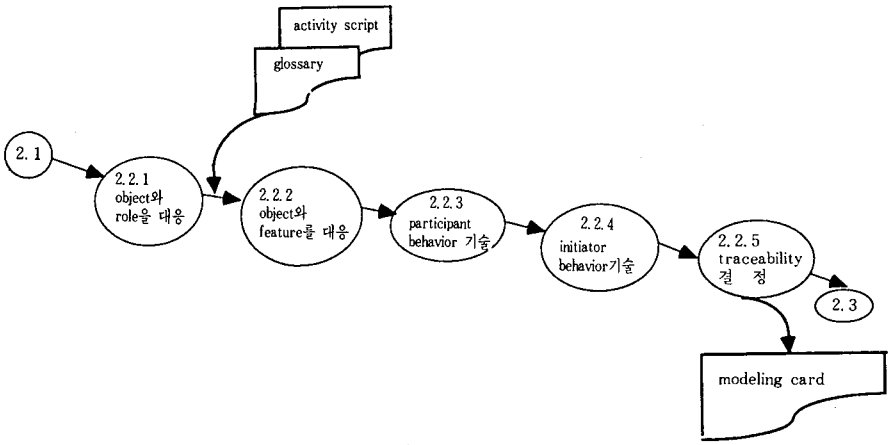


그림 7. Object 추출

V. 再使用 시스템

소프트웨어의 再使用 시스템은 Ada, Smalltalk, Eiffel 및 C++를 이용하여 개발한 시스템들이 있다. 대개 비슷한 기능과 구조로 구성되고, 다만 user interface 개념의 도입 여부, 라이브러리의 적용범위등에서 차이가 있을 뿐이다. 따라서 본장에서는 중앙대학교 소프트웨어 공학 lab.에서 개발한 MS/DOS용 CARS 1.0과 개발중인 UNIX/X-Window용인 CARS(computer aided reuse system)2.0을 중심으로 하여 설명하겠다.

1) CARS 2.0의 구성과 subsystem

CARS 2.0은 CARS 1.0의 기능을 확장하여 UNIX와 X-Window하에서 개발한 재사용 시스템으로 4개의 서브시스템인,

- 라이브러리
- 라이브러리 관리시스템
- 부품평가 시스템
- CARS user interface

로 구성되어 있다.

라이브러리에는 CARS 1.0에서 개발한 자료구조, 그래픽스, 윈도우 및 커널을 확장한 이외에도 몇개의 응용 영역별로 라이브러리를 구축하는 예를 포함시킬 것이다. 또 부품 검색을 지원할 수 있는 부품의 속성과 term dictionary를 추가할 것이다. 라이브러리 관리 시스템은 DBMS의 기능을 갖추고 부품의 등록, 재사용을 위한 정보 제공과 정보 관리를 할 수 있도록 개발하고 있다. 부품평가 시스템은 부품을 라이브러리에 등록하기 전에 품질 평가를 하고, 재구성된 소프트웨어도 품질평가 할 수 있게 된다.

CARS user interface는 CARS 라이브러리를 사용하기 위한 검색 지원기능, 설계지원 기능, building 기능 및 새로운 부품의 추가로 달라지는 inheritance에 의한 부품 구조를 변경하는 restructuring 기능을 포함하게 된다. 그리고 CARS user interface를 활용하여 특정한 사용자의 영역을 대상으로 한 application user interface를 객체 지향 방법으로 개발할 수 있도록 유도하고 있다. 다음은 4가지 서브시스템의 주요 기능을 나열하였다.

(1) Library
- 영역별 library
- 부품의 속성
- Term dictionary

(2) Library management system
- 부품등록
- 재사용 정보제공
- 재사용 정보 관리

(3) 부품평가 시스템
- 크기
- 문서화 정도
- 코드의 표준화
- 이식성의 기준
- Cohesion
- Coupling
- Complexity

(4) CARS user interface
- 검색지원 기능
  - ①질의어
  - ②평가지원 툴
  - ③검색지원 툴
- 설계 지원 기능
  - ①객체 지향 모델링
  - ②객체 지향 분석 및 설계
  - ③구현
  - ④Browsing 툴
- Building 툴
- Restructuring 툴

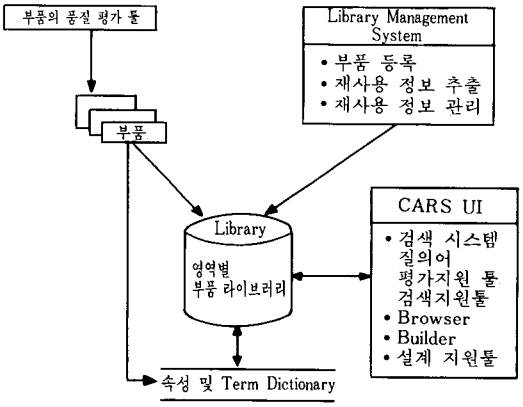


그림 9. CARS 2.0의 구성

2) 부품의 특성

CARS의 생산적인 활동을 위해서는 라이브러리에 저장한 부품을 작성하는 기술이 제일 중요하다. 정보모델링부터 요구 정의를 하고 객체(object)를 설계하

는 기술은 별도의 문헌과 교육 훈련을 통해서 이루어지겠고, 여기에서는 객체를 C++로 코딩하는 방법을 간단히 소개한다.

부품은 명세 부분과 구현 부분으로 나누어진다. 명세 부분은 부품을 어떻게 사용할 수 있는가를 정의하며, 구현 부분은 부품을 구현하는 방법을 기술한다. 명세는 구문(syntax)과 의미(semantic)로 구분하고, 구현은 표현 기법과 알고리즘으로 구분한다. 구문은 구현된 abstrat data type을 구성하는 함수들의 이름과 함수 호출시 필요한 오퍼렌드의 수와 타입을 제공하며, 의미는 각 자료형의 가능한 모든 함수에 대해서 가능한 입력 자료와 출력 자료에 관한 정보를 기술한다. 표현 기법은 부품의 값들이 메모리상에서 어떻게 표현되는 가를 기술하는 것이며, 알고리즘은 부품의 함수를 구현하는 방법을 기술하는 것으로 표현 기법이 어떻게 사용되고 조작되는 지를 설명하는 부분이다.

3) 소프트웨어 분류방법

소프트웨어 부품을 재사용할 때 발생하는 가장 큰 문제점중의 하나가, 어떻게 하면 필요한 소프트웨어 부품을 쉽게 찾아 사용할 수 있는가 하는 것이다.

소프트웨어 부품을 재사용하는데 있어서 원하는 부품을 찾고 이해하는데 드는 시간이 새로 작성되는데 드는 시간보다 길다면 부품의 재사용을 피할 것이다. 이런 재사용 부품의 식별능력은 소프트웨어 부품 표현 방법에 큰 영향을 받는다. 즉, 재사용 부품들은 다른 부품들과 쉽게 구별될 수 있도록 그 특성들이 표현되어야 한다. 또한 사용자가 원하는 부품을 찾지 못했을 경우에는 원하는 부품과 가장 유사한 부품을 제공할 수 있어야 한다. 따라서 재사용 시스템을 설계하는데 가장 기본적으로 고려해야 할 사항이 바로 부품의 분류인 것이다.

분류란, "어떤 영역 안에서 비슷한 성질들의 부품들을 그들의 특성에 따라 일정한 방법으로 모아 놓은 것"이라고 할 수 있다. 그러므로 각 부품들은 특성에 따라 모아지게 되고, 모아진 각각의 부품들은 다른 그룹의 부품들이 가지지 않는 공동의 특성을 갖게 된다. 부품들은 소프트웨어 라이브러리에 저장되는데, 효과적으로 분류되고 표현되어 저장되어야 한다. 또한 사용자가 쉽게 부품을 찾을 수 있도록 체계적으로 구성되어야 하며, 새로운 부품을 쉽게 확장 시킬 수 있는 확장성도 있어야 한다. 지금까지 알려진 분류 방법으로는 enumerative 분류방법, faceted 분류 방법, 구조-관계적(structured-relational) 분류 방법등이 있다.

CARS 2.0의 부품 분류는 facet 분류안을 바탕으로 하고, 하나의 facet내에서 관련성에 관한 정보를 별도로 관리할 수 있는 구조를 갖는다. 따라서 두개의 메카니즘의 조합에 의해서 분류 전략을 세우고 있다. 첫번째 메카니즘은, 라이브러리에 추가될 각 부품에 계층적 분류 코드를 할당하는 것으로 분류 코드는 라이브러리의 형태를 나타낸다. 예를 들면 자료 구조 코드, 수학적 함수, 검색 코드등으로 라이브러리의 형태를 구분할 수 있다. 두번째 메카니즘은, 각 라이브러리의 특성을 반영하는 기술적 키워드 즉, facet을 사용하는 것으로 이 키워드는 분류 코드와 중복이 가능하며 확장이 가능하다. 예를 들면 자료 구조에 대한 라이브러리 구축시는 각 자료 구조의 특성인〈접근법, 순회, 방법, 저장소〉로 facet을 선정하고 term을 결정한다. 그러나, 개발된 또 다른 라이브러리인 그래픽 라이브러리의 경우〈기능, video 특성, 적용 영역, 적용대상〉으로 facet을 선정하고 term을 결정할 수도 있다. 또한 이러한 facet들 사이의 계층적 관련도도 부품의 특성을 반영하여 작성, 관리하게 된다.

Ⅵ. 결 론

Object modelling은 Yourdon이 제안한 분석과 설계 과정을 구별하여야 하는 방법보다도 여기에서 제안한 modelling card 기술방법이 再使用 시스템에 적용하기에 더욱 편리하다. Object modelling이 응용영역별로 다양하게 개발된다면 再使用의 적용은 대단히 넓어질 것이다.

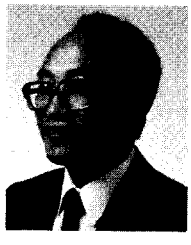
더욱이 C++언어의 결점인 interactive, incremental compilation, 자동메모리 관리, method combination 및 dynamic redefinition등의 기술이 개발되어 새로운 객체지향언어가 개발되고 UIMS 기술이 발전되면 소프트웨어 自動化가 구체화되어 갈 수 있을 것이다.

소프트웨어의 自動 生産단계에 이르기 위해서는 object relation을 정의하여 部品分類(classification) 기술을 확립하기 위해서 대수학의 catagory 이론을 적용할 수 있는 연구가 꾸준하게 진행되어야 한다.

參 考 文 獻

[1] 이경환, 소프트웨어 再使用시스템 CARS 1.0 메뉴얼, 1991.
[2] 이경환, 소프트웨어 재이용에 관한 연구, 소프트웨어 공학연구회 '91학술발표논문집, 1991.
[3] R. G. Lanergan, B. A. Poynton, Reusable Code, IBM User Group, 1979.
[4] Y. Matsumoto, A Software factory, Tutorial, Software Reusability, pp. 155-178, Dec. 1986.

筆 者 紹 介



李 京 煥
1940年 10月 7日生
1964年 중앙대학교 수학과(이학사)
1966年 중앙대학교 대학원 수학과(이학석사)
1980年 중앙대학교 대학원 응용수학 전공(이학박사)

1972年~현재 중앙대학교 전자계산학과 교수, 전자계산 소장
주관심분야 : 소프트웨어 공학, Object-Oriented Paradigm 재사용