설계모듈의 분류방법에 관한 연구

변상용*, 안기중*

A Study on the Classification of the SW Design Module

*Byun Sang-yong**, *Ahn Khi-jung**

Summary

The software reuse enhances software qualities and development productivities. As basic approaches for software reuse, acquiring, understanding, modifying, and integrating of software components has been proposed. Among of them, finding a candidate component is very important and it depends on the classification scheme.

In this paper, traditional classification methodologies on software components are reviewed and then new methodology, ENUFACE (ENUmerative-FACEted), is proposed. A searching efficiencies are analysed in both methodologies.

서 론

소프트웨어의 개발생산성을 향상시키기 위해서 많은 기법들이 개발되어 왔다. 근래에 와서는 소프트웨어의 재사용과 객체지향적 개발방법이 각광을 받고 있다. 객체지향적 접근방법은 아직 그 효용성이 완전히 입증되지 못하였고(Loy 90), 소프트웨어 재사용은 실험적으로 그 효용성이 입증되어 왔다(Free,83, Cava,83, Neig,84, Barn,87). 특히 일본에서 소프트웨어 팩토리 개념으로서 소프트웨어 재사용이 성공을 거두자(Mats 86, Nish 88), 전 세계적으로 소프트웨어의 재사용을 연구하는 실정이다.

소프트웨어 재사용은 소프트웨어 간의 유사성과, 유지보수와 재사용이 상호 밀접한 유사성이 있다는 이유로 더욱 관심이 집중되어 왔다(Trac 88). 여러 영역의 프로그램을 분석한 결과, 많은 프로그램들이 40~60% 이상의 유사한 기능을 가짐으로, 이러한 기능들을 재사용할 수 있다고 보고하고 있다(Kang 87).

소프트웨어 재사용 기법을 이용하여 소프트웨어를 개발한다면, 소프트웨어 개발 생산성이 향상될 것이고, 일치성, 관리성, 표준화 등의 많은 이득을 얻을 수 있다고 한다(Bigg 87, Trac 87). 특히 기존 모듈을 사용하기 위해서, 설계를 바꾸어 생산성과 표준화를 제고시킬 수 있다.

재사용을 위한 원칙적인 방법으로서, 원하는 부품의 획득, 부품의 이해, 부품의 수정, 부품의 결합 등의 면을 제시하고 있다(Bigg 87, Prie 87,

* 공과대학 정보공학과

Agre 88). 여기에서는 설계단계에서 발생하는 모듈의 재사용을 위해서, 모듈의 분류와 검색에 대해 논한다. 모듈은, 하드웨어의 설계에서 블랙박스로 처리하는, 전자칩에 비교할 수 있고(Lenz 87), 개발단계에서는 모듈의 입출력 자료, 모듈의 기능만 알면 된다(Wald 87). 모듈의 재사용률은 응용영역이 유사할 수록 높고, 많이 개발되는 소프트웨어가 사무처리분야이기 때문에, 이 영역에서의 모듈분류로 한정한다.

모듈의 분류

1. 모듈 분류의 필요성과 분류 조건

소프트웨어 부품을 재사용할 때 발생하는 가장 큰 문제점 중의 하나가, 어떻게 하면 필요한 소프트웨어 부품을 쉽게 찾아 사용할 수 있는가 하는 것이다. 마치 어떤 하드웨어를 개발할 때, 설계자가 그 하드웨어 구성에 필요한 부품들을, 각 회사에서 제공하는 물품목록을 보고서, 그 부품의 기능이나 사용방법 등을 잘 인식하여 선택하듯이, 소프트웨어 부품의 기능이나 특성을 정형화된 방법으로 인식하여 쉽게 재사용하도록 하여야 한다. 소프트웨어 부품을 재사용하는데 있어서 원하는 부품을 찾고 이해하는데 드는 시간이 새로 작성하는데 드는 시간 보다 길다면 부품의 재사용을 회피할 것이다.

따라서 재사용 시스템을 설계하는데 가장 기본적으로 고려해야 할 사항이 바로 부품의 분류인 것이다. 분류란 "어떤 영역안에서 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, 부품들은 효과적으로 분류되고 표현되어 저장되어야 하고, 사용자가 부품을 쉽게 찾을 수 있도록 체계적으로 구성되어야 하며, 새로운 부품을 쉽게 추가시킬 수 있는 확장성도 있어야 한다.

그러므로 부품을 분류할 때는 다음과 같은 사항을 고려하여야 한다(Prie 87).

- 원하는 부품을 찾기에 용이할 것
- 새로운 특성을 갖는 부품이 추가될 때 확장이 용이할 것
- 원하는 부품을 찾지 못한 경우에도, 유사한 부품을 제시할 수 있어야 함
- 유사한 부품이 많은 경우에도, 쉽게 찾을 수 있을 것

지금까지 사용되는 분류방법에는, 계층적(hierarchical 또는 enumerative) 분류방법, faceted 분류방법, 구조-관계적(structured-relational) 분류방법이 있다. 먼저 이들에 대해서 논한다음, 본 논문에서 제안하는 ENUFACE (ENUmerative-FACEted) 분류방법에 대해서 논한다.

2. Enumerative 분류방법

이 분류방법은 전통적인 방법으로서, 지식전체가 모두 합성가능한 클래스들을 포함할 수 있는 것으로, 점차로 좁은 클래스들로 나누어질 수 있고, 그들 사이의 계층적인 관련성 표현에 그 특성을 두고 있다. 이 방법은 계층관계 속에 모든 가능한 서브클래스와 합성된 클래스들을 포함하고 있다(Prie 87). Fig.1은 듀이 분류의 일부분을 예로 보인 것이다(Prie 87). 이 그림은 계층적인 특성을 트리구조로 나타낸다. 여기에서 최하위 노드가 실체에 해당하고, 중상위의 노드들은 실체들을 대표하는 추상적인 개념이라고 볼 수 있다. 따라서 어떠한 요소를 찾을 때는, 트리의 깊이 우선 검색을 따른다.

이러한 분류방법은 요소들간의 상호관계를 잘 나타내기 때문에, 다른 분류방법에 비해, 원하는 요소를 신속하게 찾을 수 있다는 장점이 있다. 하지만 분류된 것들과 성질을 달리하는 새로운 요소를 추가할 때는, 분류계층을 다시 해야 하는 어려움이 있다. 즉 확장성이 미진하다. Fig.1에서, "구조적 시스템 프로그래밍"이라는 제목이 미리 정의된 클래스의 어느 하나에 적합하게 나타낼 수 없다.

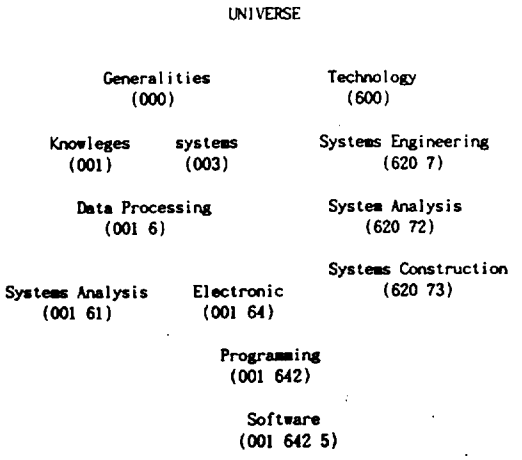

Fig. 1. Example of Dewey's decimal classification

3. Faceted 분류방법

이 분류방법은 enumerative 분류방법의 단점을 개선하기 위해 등장한 방법이다. 이 방법은 부품들이 가지는 공통적인 측면의 속성들을 하나의 facet으로 표현하는 방식으로, 한 부품을 여러 facet으로 나타낼 수 있다. 따라서 facet을 견해, 관점, 특별한 영역의 차원으로 생각할 수 있다 (Prie 87). 그리고 각 facet은 그를 구성하는 원소인 항목을 가진다. Fig.2는 이 분류방법에 대한 도형적 표현이다(Lee 90). 그림에서 알 수 있듯이, 하나의 요소를 검색할려면, 여러 facet에서 관련된 항목을 추출하여 합성하여야 한다. Fig.3은 Fig.1에 나타난 내용을 이 분류방법으로 나타낸 예이다.

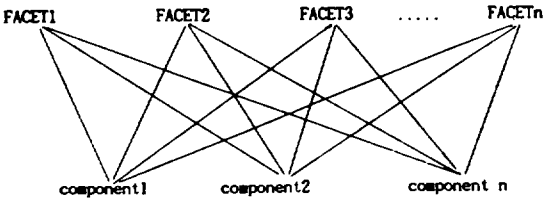

Fig. 2. Faceted classification

이 방법의 장점은 요소를 추가하기 쉽기 때문에 확장성을 지원한다. "구조적 시스템 프로그래밍"을 분류할려면, (…system programming…)이라는 종합된 클래스를 만들어 내면된다. 하지만 facet의 항목이 많아질 때, 그들 사이의 관련성을 명세하기가 어렵고, 동의어 처리가 어렵다. 특히 원하는 요소를 찾기위해서 browsing 기법을 도입했을 때, 검색시간이 너무 길어진다는 단점이 있다.

....	entity	activity	.....
	1 programs	1 programming	
	2 designs	2 design	
	3 systems	3 evaluation	
	4 structure	4 analysis	
	.	.	
	.	.	
	.	.	

Fig. 3. Faceted classification for Fig. 1

4. Structured-relational 분류방법

이 분류방법은 앞에서 기술한 두 분류방법의 단점은 보완하고 장점을 취한 방법이다. 이 방법은 계층적 조직의 찾아보기(browsing) 역량과 관계적 방법의 유동성과 재조직의 용이성을 제공한다. 이 분류방법에는 두 가지 중요한 부분이 있다. 즉 Fig.4에서 나타난 바와 같이, 관계적인 부분과 구조적인 부분이다(Lee 90). 관계적인 부분(그림에서 평행사변형)은 부품과 그들의 속성에 대한 표에 의해 제공된다. 구조적인 부분(그림에서 여러 가지 트리와 격자구조들)은 속성들의 값의 영역이 그 영역의 의미로 부터 나오는 부가적인 구조를 갖는다고 가정함으로써 생긴다. 이러한 구조들은 격자, 연결리스트, 네크워크 등일 수 있다. 이러한 것은 검색 시에 두 가지 다른 행위를 이끈다. 하나는 특별한 속성의 구조적 값을 통한 검색이고, 다른 하나는 속성 값이 주어진 값의 집합과 (부분적으로) 일치하는 모든 부품을 선택하는 것이다.

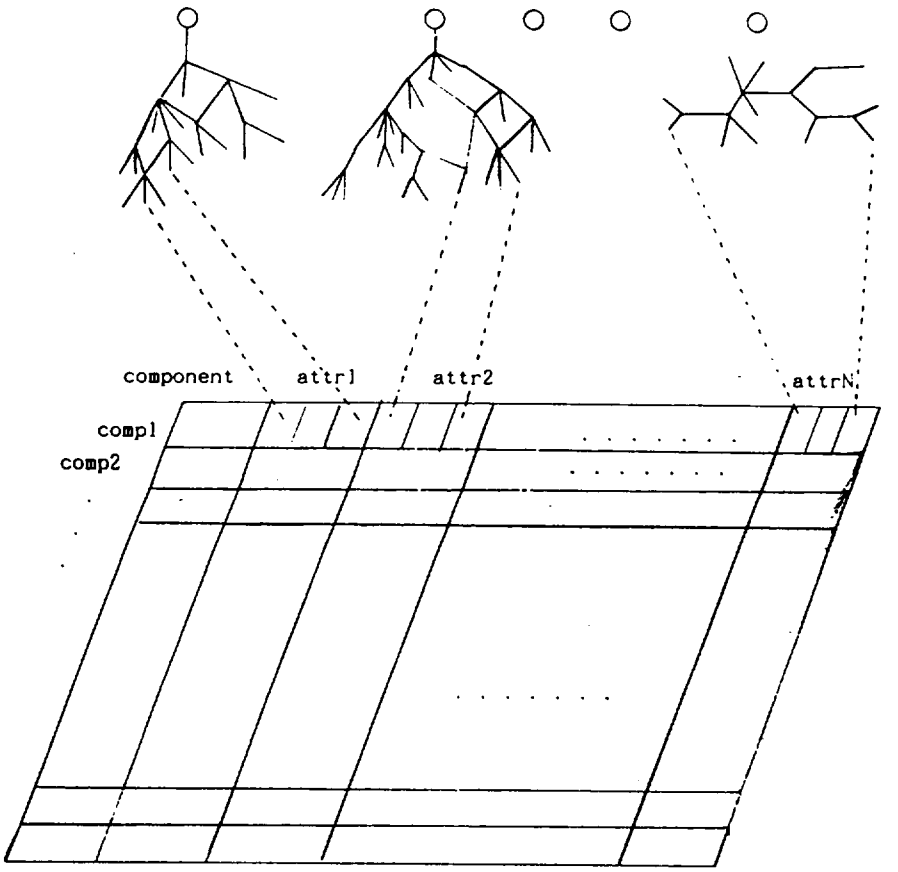

Fig. 4. Structured-relational classification

5. ENFACE (ENUmerative-FACTted) 분류방법

(1) ENUFACE 분류방법의 배경

모듈의 재사용은 하드웨어에서의 전자칩의 역할과 매우 유사하게 볼 수 있다. 하드웨어 설계자는 하드웨어를 설계하기 이전에 필요한 전자칩을 먼저 생각한다. 이것은 각 회사의 물품목록을 보면서, 각 전자칩의 입력, 기능, 출력을 보고 결정할 수 있다. 그리고나서 준비된 부품을 이용하는 설계를 한다(Rice 83, Lenz 87). 이와 마찬가지로 소프트웨어 설계자는 소프트웨어를 설계하기 전에, 우선 필요한 부품을 찾게 될 것이다. 이 때, 모듈의 입력, 기능, 출력을 알면 해당되는 모듈을 선택하여 사용할 수 있게 된다.

따라서 모듈을 사용하고자 할 때에는 반드시 모듈의 입력, 기능, 출력을 알아야만 한다. 계층적 분류방법에서 이러한 정보를 한꺼번에 표현하기란 거의 불가능하다. faceted 방법에서는 이와 같은 사실을 쉽게 표현할 수 있다. 즉, 입력과 기능, 출력이라는 facet을 이용함으로써 간단히 처리할 수 있다. 구조-관계적 방법에서는, 관계적인 방식으로는 가능하지만, 구조적인 방식으로는 계층적 방식과 유사하므로 거의 불가능하다. 결국 모듈의 재사용에서 모듈에 대한 입력과 기능, 출력을 한꺼번에 알아야만, 모듈을 사용할 수 있다는 가정 하에서는, faceted 방법이나 구조-관계적 방법이나 유사하다고 볼 수 있다.

Faceted 방법은 facet에서의 항목을 키보드로

부터 받아들일 경우에는, 동의어사전 개념적 거리 그래프 등의 복잡한 접근방법이 필요하다. Browsing 방법을 사용하면, 검색시간이 많아진다.

따라서 본 논문에서는, 기본적으로는 faceted 방법을 따르면서, 검색시간의 단축을 위해 계층적 방법을 혼합한 ENUFACE 방법을 제안한다.

(2) ENUFACE 분류방법의 내용

이 방법은 Fig.5에서와 같이, 기본적으로는 〈응용분야, 기능, 입력, 출력〉이라는 facet을 사용하여 모듈을 표현한다. 응용분야가 필요한 이유는, 같은 입력, 기능, 출력을 가진 모듈이라도 응용분야에 따라서는 다른 모듈이 존재할 수 있고, 검색에 유용하기 때문이다. 검색할 때는 Fig.6과 같이 기능그룹 수준이 더 꺼어들게 된다. 이 그림 이 계층적 방법과 유사하나, 깊이가 3 밖에 되지 않는다는 것이 다른 점이다.

Applicaion	function	input	output
A1	f1	i1: i2: i3	o1: o2
A1	f2	i4: i5	o3
A2	f1	i6	o4
A2	f3	i7: i8	o5
.	.	.	.
.	.	.	.
.	.	.	.

Fig.5. 모듈의 faceted 분류

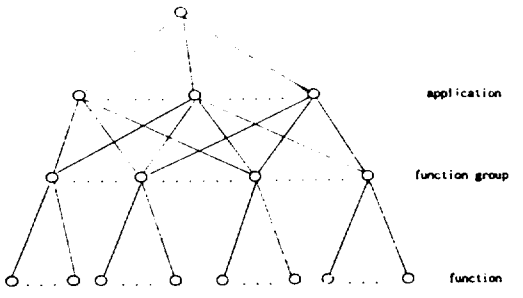

Fig.6. Hierarchical search of modules

코볼 프로그램 영역에서의 한 응용프로그램의 실행가능한 원시코드를 20,000 라인이라고 가정하자. 이러한 크기의 프로그램은 소규모와 대규모 프로젝트의 기준점이 된다. Lanergan & Grasso가 5274개의 프로그램을 대상으로 한 연구에 의하면, 한 프로그램의 평균 코드는 프로그램의 성격에 따라 500~800라인을 갖는다(Lane 84). 이것으로 볼 때 상기의 가정은 상당한 악조건이라고 할 수 있다. 한 모듈은 20~50라인으로 작성하는 것이 효율적이다. 따라서 상기의 응용프로그램은 40~100개의 모듈을 갖는다. 한 프로그램이 100개의 모듈을 가질 때, 이러한 모듈의 이름은 기능 facet의 항목으로 나타난다. 하지만 한 화면에는 20개 이상의 모듈을 나타낼 수 없으므로 이러한 항목들은 5개 이상의 화면으로 나누어 나타내야 하는데, 이렇게 되면 사용자에게 미치는 불편이 커지게 된다.

따라서 100개의 모듈을 20개 이하의 그룹으로 나눌 필요가 있다. 이와 같은 그룹을 기능그룹 facet으로 나타낸다. 기능그룹 수는 5(100/20) 이상이면 된다. 본 논문에서는 다음 절에 언급하는 연구보고를 바탕으로 기능그룹 수를 9로 하였다. 기능그룹 수가 9이기 때문에 한 화면에 나타낼 수 있다. 계층분류의 깊이는 응용분야의 수에 따라 결정된다. 본 논문에서는 계층분류의 깊이를 3이라고 한다.

(3) 기능그룹의 분류

응용분야는 모듈이 사용되는 분야이고, 기능그룹은 여러 모듈의 기능들을 특성에 따라 모은 것이다. T.C. Jones의 연구에 의하면, 코볼 프로그램의 기능은 계산, 기호적 논리, 조건에 대해, 정렬, 합병, 변환, 자료선언, 자료입력 등으로 나타낼 수 있다고 보고한다(Jone 84). Arthur는 편집 및 자료확인, 갱신, 테이블 처리, 보고서 작성, 온-라인, 보안, 통합으로 나타낼 수 있다고 보고한다(Arth 85). 또 일본 정보처리 진흥사업협회의 보고에 의하면, 코볼 프로그램의 기능을 테이블 처리, 헤드체크, 헤드처리, 날짜와 시간, 코드변환, 계산 등으로 나타낼 수 있다고 한다(日本情報 86). 또한 우리나라에서 발표한 연구보고는 입력,

출력, 자료이동, 테이블 검색, 계산 등으로 코볼 프로그램 기능을 나눌 수 있다고 한다(안상형 88). 상기 연구 결과와, 81명의 코볼 프로그램 실무자들에게 설문조사를 한 결과를 바탕으로, 사무처리 영역의 기능을 다음과 같이 분류한다.

- 입력 : 자료의 입력과 자료에 대한 오류검사 등을 수행하는 모듈들의 집합
- 출력 : 자료의 출력을 수행하는 모듈들의 집합
- 계산 및 처리 : 매개변수를 입력으로 받아 출력을 만들기 위해 일련의 처리를 하는 모듈들의 집합
- 정렬 : 자료를 받아들여 순서에 따라 정렬하는 모듈들의 집합
- 합병 : 둘 이상의 복합자료를 받아들여 합병을 하는 모듈들의 집합
- 변환 : 자료를 받아들여 다른 자료로 변환을 하는 모듈들의 집합
- 검색 : 주어진 테이블 또는 화일 등에서 주어진 항목을 찾는 모듈들의 집합
- 갱신 : 주어진 테이블 또는 화일 등에서 내용에 대한 갱신을 처리하는 모듈들의 집합
- 기타 : 상기 이외의 모듈들의 집합

(4) ENUFACE 분류에서의 검색과정

검색은 browisng 기법을 도입한다. 이것은 사용자의 편리를 위한 것이다. 모듈을 사용할려면, 모듈의 입력, 기능, 출력을 알아야만 한다. 이것이 검색을 위한 기본 facet이 되고, 기능을 찾을 때는 계층적인 분류를 이용하도록 하는 것이 본 논문에서 추구하는 중요한 것이다. 계층은 응용분야, 기능그룹, 기능으로 나누어 진다. 응용분야 facet에서 하나의 항목을 선택하면, 그 응용분야에 따른 기능그룹이 나타난다. 여기에서 하나의 항목을 선택하면, 그 기능그룹에 해당하는 기능들이 나타나도록 한다. 이렇게 함으로써, 선택이 이루어질 때 마다 검색 공간을 줄여나갈 수 있다. (Fig.7 참조.)

이렇게 하여 기능을 찾게 되면, 그 기능에 해당되는 입력자료와 출력자료가 입력과 출력 facet에 나타난다. 여기에서 원하는 입력자료와 출력자료를 선택하면, 〈응용분야, 기능, 입력, 출력〉이라는 합성된 값에 의해서 질의가 형성되고, 재사용 모듈 라이브러리에서 해당 모듈을 검색한다.

검색효율에 대한 평가

앞에서 본 바와 같이, 분류방법론은 여러 가지가 있다. 그러나 설계정보의 재사용을 위한 방법론으로서 적합한 것은 faceted 방법, 구조-관계적(structured-relational) 방법과 본 논문에서 주장하는 ENUFACE 방법이다. 이 중에서, 구조-

응용분야	기능그룹	기능	입력자료	출력자료
*	*	*	*	*
인사관리	입력	시간외 수당 계산	없 음	없 음
급여관리	출력	세금계산	일과외 근무시간	시간외 수당
회계관리	계산및 처리	공제액 계산	야간 근무시간	.
재고관리	갱신	.	휴일 근무시간	.
금융관리	.	.	휴일 야간 근무시간	.
.	.	.	.	
.	.	.	.	
.	.	.		
.	.	.		

FACET : < 급여관리, 시간외 수당 계산, (일과외 근무시간: 야간 근무시간: 휴일 근무시간: 휴일 야간 근무시간), 시간외 수당 >

Fig.7. Search of module

관계적 방법은 faceted 방법과 유사하게 볼 수 있다. 따라서 본 논문에서는, faceted 방법과 ENUFACE 방법의 비교로 제한한다. 또, 두 방법 모두 browsing 방식을 사용하는 것으로 하고, 모듈을 찾을 수 있는 질의를 만드는데 걸리는 시간을 측정하고 비교한다. 이하에서의 검색시간은 엄밀하게는 질의어 생성시간을 말한다. 하지만 질의어 생성시간 이외에, 실제 모듈검색을 위한, 프로그램 자체의 수행시간은 질의어 생성시간에 비하면 무시해도 될 정도로 아주 짧기 때문에 평가의 편리를 위해서 생략한 것이다.

1. Faceted 방법에서의 평균검색시간 측정

faceted 방법에서의 평균검색시간(Ft)은 식(1)과 같이 나타낼 수 있다.

$$Ft = \frac{(Na+Nm+Nin+Nout)*t}{2} \quad \cdots\cdots (1)$$

Na : 응용분야 facet에 나타나는 항목 수
Nm : 기능 facet에 나타나는 항목 수
Nin : 입력자료 facet에 나타나는 항목 수
Nout : 출력자료 facet에 나타나는 항목 수
t : 각 facet에서 한 항목을 살펴보는 시간

식(1)은, 한 facet에 있는 항목을 선형적으로 살펴보는(linear search) 평균시간을(항목수/2) * 항목당 소요시간으로 한다. 엄밀하게는 (항목수+1)/2 *항목당 소요시간이지만 계산의 편의를 위하여 '+1'은 생략한다. 응용분야, 기능, 입력, 출력의 4개의 facet이 존재하기 때문에, 각 facet에서의 항목선택에 소비되는 평균시간들을 합하여, 전체의 평균시간으로 한 것이다. 식(1)은 다시 식(2)와 같이 나타낼 수 있다.

$$Ft = \frac{Na+(Nm+\{(1-Ri)*Nm*Ni\}+\{(1-Ro)*Nm*No\})*(1-Rm)}{1-Rm} * t \quad \cdots\cdots (2)$$

Na : 응용분야 수
Nm : 모듈 수
Rm : 중복률 (프로그램간 모듈 중복률)
Ni : 평균 입력자료 수
No : 평균 출력자료 수
Ri : 모듈간 입력자료 중복률
Ro : 모듈간 출력자료 중복률
t : 항목당 소요시간

식 (2)에서, (1-Rm)은 모듈간의 중복을 배제함을 의미한다. 즉, 전체 응용분야를 살펴보면 Nm개의 모듈 수가 존재하지만, 실제 기능 facet에 포함되는 항목들은 응용분야간 중복되는 것은 한 번만 나타내면 된다는 것을 의미한다. 그리고 (1-Ri)와 (1-Ro)는 각각 모듈간 입출력자료의 중복을 배제함을 의미한다. 즉 전체적으로 살펴보면, 응용분야간, 모듈간 중복되어 나타나는 입출력 자료가 존재하기 때문에, 모듈간, 입출력 자료간 중복률을 사용하여 중복을 배제하고 있다. 식 (2)에서는 모듈당 평균 입출력 자료수를 이용하여 입출력 자료수를 계산하고 있는데, 이것은 ENUFACE와의 비교를 편리하게 하기 위해서이다.

2. ENUFACE 방법에서의 평균검색시간 측정

ENUFACE 방법에서의 평균검색시간(Et)은 식(3)과 같이 나타낼 수 있다. 이 식도 앞 절의 개념이 그대로 적용된다.

$$Et = \frac{Na+Nf+Nm+Nin+Nout}{2} * t \quad \cdots\cdots (3)$$

Na : 응용분야 facet에 나타나는 항목 수
Nf : 한 응용분야에 나타나는 평균 기능그룹 수
Nm : 기능그룹에 해당하는 모듈 수
Nin : 모듈에 해당하는 입력자료 수
Nout : 모듈에 해당하는 출력자료 수
t : 각 facet에서 한 항목을 살펴보는 시간

식 (3)에서, 그룹수는 앞에서 언급한 바와 같이

입력, 출력, 계산 등의 기능그룹의 평균갯수이다. 모듈수는 전체 모듈의 갯수가 아니라, 한 기능그룹에 속하는 모듈의 갯수이다. 입출력자료 역시 한 모듈에만 속하는 입출력 자료의 갯수이다.

식 (3)은 다시 식 (4)와 같이 나타낼 수 있다.

$$Et=\frac{Na+Nf+(Nm/(Na*Nf)+Ni+No}{2}*t\cdots(4)$$

Na : 응용분야 수
Nf : 평균 기능그룹 수
Nm : 전체 모듈 수
Ni : 모듈당 평균 입력자료 수
No : 모듈당 평균 출력자료 수
t : 항목당 소요시간

4. 두 방법론의 종합비교

분석의 편의를 위해, 다음과 같이 가정한다.

Na=20, Nm=1000, Rm=(0.4-0.6), Nf=9,
Ni=3, No=1.5, Ri=(0.05-0.5), Ro=(0.05-0.4)

응용분야수와 모듈수는 비교를 위한 것이고, 모듈중복률은 소프트웨어간 기능의 유사성을 근거로 하였으며, 평균 기능그룹 수는 앞절의 연구를 참고하여 가정하였으며, 입출력 자료수는 보통의 모듈이 갖는 입력 수는 0-5, 출력 수는 0-3 정도라는 일상감각에서 근거한 것이다. 또 모듈간 입출력 자료 중복률은 재사용 모듈을 충분히 준비하기 어려운 상황에서 이론적으로 평가하기 위한 가정이다.

상기와 같이 가정을 했을 때, faceted 방법에서의 평균검색시간은 Table 1과 같다. Table 1에서 살펴보면, 모듈중복성이 50%이고, 입출력 자료의 중복률이 각각 20%일 때, 평균검색시간은 960t라는 것을 알 수 있다. 여기서 한 가지 특기할 것은, 만일 faceted 방법을 사용하지 않고, 모듈을 직접 하나하나 살펴보는 원시적인 방법을 사용한다면, 평균검색시간이(250t+alpha)면 될 것이라는 사실이다. 왜냐하면, 모듈중복성이 50%이기 때문에 실제적인 모듈수는 500개이고, 이에 대한

Table 1. Average search time in faceted classification

(unit : 10t)

	40% redundent					50% redundent					60% redendent				
I/O	5	10	20	30	40	5	10	20	30	40	5	10	20	30	40
5	131	129	124	120	115	109	107	104	100	96	88	86	83	80	77
10	128	126	121	117	112	107	105	101	97	94	86	84	81	78	75
20	122	120	115	111	106	102	100	96	92	89	82	80	77	74	71
30	116	114	109	105	100	97	95	91	87	84	78	76	73	70	67
40	110	108	103	99	94	92	90	86	82	79	74	72	69	66	63

* I/O : input/output redundent rate

평균검색시간은 250t면 된다. 하지만 이 방법을 사용할 때는 원하는 모듈과 유사한 모듈의 입출력에 대한 사항도 같이 살펴보아야 하기 때문에 alpha 만큼을 더 포함시켰다. 일상적인 감각으로 이야기한다면, faceted 방법이 원시적인 방법보다 검색시간이 더 걸릴 가능성이 많다. 또한 faceted 방법에서 문제가 되는 것은, facet의 항목이 많아짐에 따라, 화면의 scroll-up과 scroll-down이 빈번해 지기 때문에 사용자에게 큰 불편을 초래하여, 모듈의 재사용을 포기할런지도 모른다.

한편 ENUFACE 방법에서는 기능그룹 수의 변화는 매우 크지 않다면 검색시간에 미미한 변화만 보일 뿐이다. ENUFACE 방법이 모듈을 찾을 때는 계층적으로 검색이 이루어지기 때문에, 모듈중복성이 검색에 영향을 미치는 정도가 미미하다. 모듈간 입출력 자료의 중복을 제외시킨 것은, ENUFACE 방법은 전체의 입출력 자료가 facet에 나타나는 것이 아니라, 한 모듈에 대한 입출력 자료만이 항목으로 나타나기 때문이다.

ENUFACE 방법에서의 평균검색시간은 19.03t이다. 이것은, 응용분야를 선택하는 평균시간이 10t, 기능그룹항목을 선택하는 평균시간이 4.5t, 모듈항목을 선택하는 평균시간 {1000/(20*9)}/2 *t=2.78t, 입력과 출력 항목을 선택하는 평균시간 1t, 0.75t를 합산한 것이다.

이상에서 faceted 방법과 ENUFACE 방법에서의 평균검색시간을 측정하였다. 기존의 가정에 따라 평균검색시간을 측정한 결과, faceted 방법에서 가장 좋은 경우의 평균검색시간이 630t인 반면

에, ENUFACE 방법에서는 19.03t로 나타났다. ENUFACE 방법을 사용하면, faceted 방법의 가장 좋은 경우에 비해, 무려 33.11배나 효율이 증가하고 있다.

Table 2는 faceted 방법에서의 최악의 검색시간을 나타내고 있다. 최선의 검색시간은 두 방법이 비슷한 값이 나오기 때문에 생략한다. 최악의 검색시간이라는 것은 검색하는 동안에 선택할 항목이 각 facet의 마지막에서 나오는 경우에 걸리는 검색시간이다. 단, 한 차례의 시도로 원하는 모듈을 찾는다는 것을 가정하고, 앞절에서 했던 가정은 그대로 이용한다. 한편, ENUFACE 방법에서의 최악의 검색시간은 각 기능그룹에 속하는 모듈의 수 중에서 가장 큰 수를 알지 못하기 때문에 정확한 검색시간을 구할 수 없다. 그래서, 기능그룹에 속하는 모듈의 수를 40, 모듈의 입출력 자료 수를 각각 5와 4로, 충분히 악조건을 가정해보자. 이 경우의 최악의 검색시간은 (20+20+40+5+4)=69t이다. 이것은 faceted 방법의 어느 경우보다 훨씬 우수하다.

Fig.8은, 다음과 같은 가정하에서, 모듈의 수가 100에서 1000까지 변할 때, 두 방법론에 대한 평균검색시간을 나타낸 것이다.

응용분야 수=20,
중복률=50%, 평균 기능그룹 수=9
평균 입력자료 수=2,
평균 출력자료 수=1.5,
모듈간 입력자료 중복률=20%,
모듈간 출력자료 중복률=5%,

Table 2. Worst search time in faceted classification

(unit : 10t)

	40% redundent					50% redundent					60% redundent				
I/O	5	10	20	30	40	5	10	20	30	40	5	10	20	30	40
5	262	257	248	239	230	218	215	207	200	192	175	172	166	160	154
10	256	251	242	233	224	213	210	202	195	187	171	168	162	156	150
20	244	239	230	221	212	203	200	192	185	177	163	160	154	148	142
30	232	227	218	209	200	193	190	182	175	167	155	152	146	140	134
40	220	215	206	197	188	183	180	172	165	157	147	144	138	132	126

* I/O : input/output redundent rate

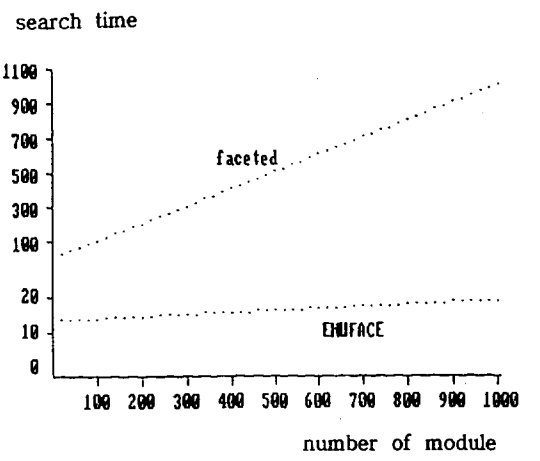

Fig. 8. Compared average search times in two method

모듈간 출력자료의 중복률을 모듈간 입력자료 중복률 보다 적게 가정한 것은, 일반적으로 모듈의 성격상 출력자료 보다는 입력자료가 중복이 많기 때문이다.

Fig.8에서, faceted 방법인 경우는 기울기가 거의 1이다. 이것은 모듈 수가 증감함에 따라, 평균검색시간이 거의 정비례한다는 것을 의미한다. 반면에, ENUFACE 방법에서는 기울기가 거의 0이다. 이것은 모듈 수가 증가하더라도 평균검색시간에 미치는 영향이 미소하다는 것을 의미한다.

적 요

모듈을 재사용하기 위해서는, 모듈의 분류가 필수적이다. 분류에 대한 기존의 분류방법인 enumerative, faceted, structured-relational 분류방법을 장단점을 중심으로 고찰하였으며, 본 논문에서 새로이 제안하는 ENUFACE 분류방법에 대해서 논하였다.

본 논문에서 제시한 ENUFACE 분류방법과 faceted 분류방법을, 모듈 검색을 위한 질의어 작성시간의 평균 개념을 도입하여 측정하여, 비교하고 평가하였다. 평가결과 본 논문에서 제안하는

ENUFACE 방법이 facet 방법에 비해, 월등하게 우수하다는 사실을 보였다. 특히 사용 대상이 되는 모듈 수가 증감하는데 따라, faceted 방법은 거의 정비례의 검색시간이 걸리지만, ENUFACE 방법은 거의 변화가 없다는 사실을 증명하였다.

본 논문은 응용분야 수에 따라 한계를 갖는다. 계층분류의 깊이가 3인 상태에서는, 응용분야 수가 2 화면을 넘어 가면 사용자에게 불편을 초래한다.

참 고 문 헌

Agresti, W.W. and F.C., McGarry 1988. The Minnowbrook Workshop on Software Reuse : A Summary Report, *Tutorial : Software Reuse : Emerging Technology*, 33~40.

Arnold, S.P. and S.L., Stepowary 1987. The Reuse System : Cataloging and Retrieval of Reusable Software, *Proceedings of COMPCONS' 87*, 376~379.

Arthur, L.J., 1985. Measuring Programmer Productivity and Software Quality, John Wiley and sons Inc., pp.138~142.

Barnes B., T. Durek J., Gaffney and A. Pyster, 1987. A Framework and Economic Foundation for Software Reuse, Proceedings of the Workshop on Software Reusability and Maintainability.

Biggerstaff, T. and C., Richter 1987. Reusability Framework, Assissment, and Directions *IEEE Software*, 41~49.

Cavaliere, M.J., 1983. Reusable code at the hartford insurance group, *ITT Proceedings of the Workshop on Reusability in Programming.*

Freeman, P., 1983. Reusable Software Engineering Concepts and Research Directions, *ITT Proceedings of the Workshop on Reusability in Programming*, 2~16.

Horowitz, E. and J.B., Munson 1984. An Expansive View of Reusable Software, *IEEE Transactions on Software Engineering*, 477~487.

Jones, T.C., 1984. Reusability in Programming : A Survey of the State of the Art, *IEEE Transactions on Software Engineering*, 488~494.

Kang, J.C., 1987. A reuse-Based Software Development Methodology, *Proceedings of the Workshop on Software Reusability and Maintainability.*

Lanergan, R.G. and C.A., Grasso 1984. Software Engineering with Reusable Designs and Code, IEEE Transactions on Software Engineering, 498~501.

Lee, T.J., 1990. *Software Reuse*, IBM.

Lenz, M., Schmid, H.A. and P.W., Wof 1987. Software Reuse through Building Blocks, *IEEE Software*, 34~42.

Loy, P.H., 1990. A Compariosn of Object-oriented and Structured Develoment methods, *Communications of the ACM SIGSOFT*, VOL. 15, No.1, 44~48.

Matsumoto, Y., 1986. A Software Factory : An Overall Approach to Software Production, *Tutorial : Software Reusability*, 155~178.

Neighbors, J.M., 1984. The Draco Approach to Constructing Software from Reusable Components, *IEEE Transactions on Software Engineering*, 564~574.

Nishino, H., K. Sigaki and M., Kishi 1988. ソフトウェアの部品組立生産システム : *MASCOT*, *FUITSU* 39, 2, 116~123.

Prieto-Diaz, R. and P.,Freeman 1987. A Software Classification Scheme for Reliability, *IEEE Software*, 6~16.

Prieto-Diaz, R. and G.A., Janes 1987. Breathing New Life, *GTE Journal of Science and*

Technology, 23~31.

Rice, J.R. and H.D., Schwetman 1983. Interface Issues in a Software Parts Technology, *ITT Proceedings of the Workshop on Reusability in Programming*, 2~16.

Tracz, W., 1987. Software Reuse : Motivators and Inhibitors, *Proceedings of COMPCONS'87*, 353~356.

안상형, 1988. 재사용 코드 라이브러리를 이용한 소프트웨어 제품 생산개념으로서의 응용 프로그램 개발에 관한 연구, 서울대학교 중앙교육연구전산원.

日本 情報處理振興事業協會 技術センター, 1986. ソフトウェア再利用と すの品質評價方法に關いる調査, pp.44~52.