휴리스틱 측정방법을 사용한 소프트웨어 모듈의 집단화에 관한 연구

변 정 우† · 송 영 재††

요 약

리엔지니어링에서 기존 소프트웨어 시스템의 환경변화에 따라 대부분 논리적 실행을 중심으로 집단화를 실행해왔으나 본 논문에서는 기존 소스 프로그램을 중심으로 각 모듈간의 정보공유측면에서 효율적으로 집단화할 수 있는 방안을 제안하였다.

정보공유를 이용한 관련 모듈들의 집단화를 위해서 모듈 집단간 휴리스틱 측정방법을 근간으로 본 논문에서 제안한 유사성 및 단일성 알고리즘을 이용한 측정을 한 후 그 결과를 평가하였다. 이를 통해 모듈 및 프로시져의 관련성을 중심으로 관련 모듈 및 프로시져의 정리 및 집단화를 유도할 수 있었다. 소프트웨어 시스템의 환경변화에 따른 기존 시스템을 정보공유를 중심으로 집단화함으로써 과적으로 소프트웨어 시스템을 재구축할 수 있는 방법론을 제시하였으며, 그 구현 가능성을 실제 예를 통해서 보였다.

A Study on the Clustering of Software Module Using the Heuristic Measurement

Byun, Jung Woo† · Song, Young Jae††

ABSTRACT

In the past, as the environment of the established software system changed, most Re-Engineering performed clustering on the basis of logical operation. In contrast, this paper proposes a method to perform clustering efficiently using the information sharing of each module of source programs that constitute the software.

For the clustering of related modules using the information sharing We evaluated the result after measuring the degree of clustering using similarity and uniqueness algorithm on the basis of heuristic method of measurement. Thus, we could manipulate and achieve the clustering of related modules and procedures. This paper also presents a method to reconstruct the software system efficiently through the clustering and shows the possibility of its realization through real examples.

1. 서 론

현대의 소프트웨어들은 윈도우즈환경 및 사용자 요구 등으로 인해 그 크기가 매우 크고 복잡한 형태로 구성되는 것이 대부분이다. 이러한 소프트웨어들이 오랜 생명력을 갖기 위해선 이해 및 수정이 쉽고, 집적성과 테스트가 용이해야 한다. 이러한 기본적 요건을 갖춘 소프트웨어 개발형태중 하나가 프로그램을 모듈(Modul)형태로 개발하는 것이다.[2)3)] 그러나 규모가 큰

† 정 회 원 : 경희호텔경영전문대학 교수
†† 종신회원 : 경희대학교 전자계산공학과 교수
논문접수 : 1998년 5월 25일, 심사완료 : 1998년 7월 16일

소프트웨어 시스템인 경우 사회환경의 급속한 발전 및 변화에 맞게 소프트웨어를 쉽게 변경한다는 것은 매우 어렵다.

이러한 소프트웨어의 환경변화시 시스템 전체를 이해함에 있어 작게는 모듈 상호간의 이해를 통해, 재구성시 원하는 부분만 수정하고 나머지 기존 부분들과의 문제가 없어야 한다는 것이 중요하다. 리엔지니어링 방법들중 구조적 차트, 호출그래프등을 이용하는 방법은 전통적으로 많이 사용하는 방법으로 많은 장점들이 있지만 알고리즘의 인식 및 설계인식등에 있어선 부적절한 면들도 있다.[10]

인식도구의 구축을 기초로 한 Slicing 메커니즘을 사용하는 방법은 대부분 오래 유지된 구성성분에 대해 그 유지기간동안 많은 기능성을 추가할 때 수정이나 재추상화를 하는데 유리하나 정확한 이해가 어렵다는 단점이 있다.[17]

IS(Interface Slicng)이라는 프로그램 slicing을 도입하여 수정하는 작업을 지원하기 위해 구성성분에 관한 정보를 제공하는 방법이 있으나 이 방법은 의미적, 구문적 재추상화를 제공하는 도구모음을 만들어 주어 구현시 나타나는 향상이나 보완을 무시하지 않는 특징을 갖고 있다. 의미이해 및 구문해석등 논리적 근거를 중심으로 원하는 추상화를 추출하나 이러한 방법들은 프로그래밍 중심으로 개발해온 많은 운영자들에게 논리적 접근이라는 측면을 중심으로 하고 있기 때문에 쉽게 이해하기가 어렵다.[9] 일반적으로 쉽게 접근할 수 있는 방법으로는 부분적인 I/O점검과 동시에 자료구조도와 관련된 응용 가능성을 점검하고, 사용된 명칭이해 및 그 명칭들의 사용목적을 파악한 후 프로시져를 분석하는 형태로 접근하는 방식이다. 그러나 리엔지니어링측면에서 이러한 환경변화에 따른 모듈변경등을 쉽게 처리할 수 있는 적절한 프로그래밍 언어와 도구들이 존재하지 않아 이러한 것을 명세(Specification)에 의존하여 문제를 해결하는 것보다는 프로그램단위들의 Cross reference정보를 분석해 많은 프로그램단위들의 범위를 추론과 정보공유라는 측면을 고려하는 것이 보다 쉽게 문제를 해결할 수 있는 하나의 방법이 될 수 있다.[4] Arch가 제시한 프로젝트 방법론은 설계자들이 시스템 구조를 분석하고 재구성을 통해 새로운 구조를 만들어서 이전 시스템과 일치하는 형태인지 감지할 수 있는 지적 도구를 만든 후 이를 통해 의미 있는 구성상 변화들이 변경이 불가능하게 되기전에 이전과 변화된 시스템사이에 검사 및 평가 도구의 필요성을 역설하였다.[4][5]

기존 시스템의 프로그램 코드로부터 그래픽과 텍스트 관점에서 기준이 되는 자료를 추출해 관련된 프로시져들을 모듈단위로 집단화할 수 있는 알고리즘이 필요하고, 정보은폐원칙들을 위배하는 각 프로시져들을 확인후 변경 또는 추가하려는 모듈간의 관계를 명확히 제시할 수 있다고 주장하였다.[1][2][4][5]

그러나 이러한 문제들은 다양한 용도로 사용되는 소프트웨어 시스템의 특징들 때문에 현실적으로 자동화하는 데에는 문제가 많다. 아울러 소프트웨어 시스템들의 특성을 파악하는 과정에서 설계자의 조정과정이 복잡할 수 밖에 없어 정형화형태로 구현하는 것이 매우 어렵다. 본 논문에서는 리엔지니어링 측면에서 프로그래머가 기존 소스코드를 이용해 정보공유와 관련 모듈들의 집단화를 통해 소프트웨어를 재구축할 수 있는 방법을 제안하고자 한다. 정보공유와 관련된 모듈의 집단화를 통해 모듈 집단간에 휴리스틱방법으로 유사성을 측정한 후, 각 모듈간의 유사성 및 단일성 측정을 통해 모듈 및 프로시져들을 정리하고, 환경변화에 따른 기존 소프트웨어 시스템을 재구축할 수 있는 방법론으로 실제 구현가능성을 예시하고자 한다.

2. 관련된 연구

2.1 정보의 공유

리엔지어링(Re-Engineering)은 소프트웨어 개발시 지원, 유지, 재사용하기 곤란한 코드들에 대한 부적절한 설계포착 및 개선하는 작업으로 간략히 표현한다면 환경변화시 시스템을 새로운 형태로 개편하기 위해 시스템을 검사하고 변경하는 형태로 볼 수 있다.[16] 이러한 리엔지니링 작업들은 대부분 소프트웨어 설계자가 아닌 다른 사람들이 하는 경우가 대부분이다. 이런 상황에서 운영자들이 설계자와 같은 수준에서 유지 관리한다는 것은 어렵다.

본 논문에서 모듈은 프로시져의 집합으로 개개의 이름을 가진 독립적으로 컴파일할 수 있는 단위로 정의하였다. 아울러 프로그램은 모듈들의 집합으로 정의하여 사용하였다.

리엔지니어링 측면에서 기존 소프트웨어 분석시 Bottom-up방식으로 소스코드를 접근하는데 여기서 중요한 것중 하나가 자료의 흐름일 것이다. 이러한 자료는

좁게는 프로시져간의 주고 받음으로부터 모듈간, 프로그램간 데이터의 주고받음으로 나타날 수 있다. 이를 통해 자료의 흐름이 어떻게 어떠한 변수 및 상수 등에 의해 공유되었는지를 알 수 있을 것이다. 2개의 프로시져 a와 b에 공통된 특징은 a∩b로, 각기 구별되는 특징은 a-b, b-a로 나타낼 수 있다. 여기서 프로시져간 공유를 휴리스틱(Heuristic)정보공유로 볼 수 있다. Tversky Ratio Model에서 2개의 프로시져간 설계상 정보를 공유하는지를 판단하는데 있어 유사성 기능에 기반을 두는 소프트웨어의 특징으로 정의하였다.[1] 여기서 몇 개의 동일한 단위이름을 사용했다면 그들은 공유된 의미를 갖는 설계정보로 볼 수 있으며 동일 모듈에 저장하기 좋은 후보자가 될 것이다. 각 모듈간의 휴리스틱한 측정을 위해서는 먼저 연관성 있는 모듈 설계상 정보를 공유하는 프로시져들을 집단화하고 이를 확인해야한다. 여기서 얼마나 정보공유를 하는가를 나타내는 유사성(Similarity) 측정은 시스템내 각 전역명칭(비지역 명칭)의 중요도를 상수화한 형태로 표현해왔으나 기존 소프트웨어의 사용용도에 따라 비지역명칭의 가중치가 다르기 때문에 현실적으로 적절치 않다. 이에 본 논문에서는 중요도 및 사용횟수에 따라 가중치(weight)를 부가하는 형태를 제안한다. 일단의 정보공유가 인정된 모듈과 그 구성인자들을 검사해 잘못 배치된 프로시져들의 수를 최소화할 수 있는 형태로 표현할 것이다.

2.2 휴리스틱방법을 이용한 유사성 및 단일성 측정

정보공유의 휴리스틱 측정은 프로시져 및 관련모듈이 사용하는 비지역명칭 즉 어떤 명칭이든 관계없이 2개이상의 프로시져내에 포함된 명칭에 기초하고 있다. 하나의 예로 C언어에 사용하는 비지역명칭은 프로시져, 마크로, typedef, 변수, 구조적type등이 된다. Tversky의 소프트웨어 유사성 측정을 위한 2가지 조건은 아래와 같다.[1][2][4]

2개의 프로시져나 모듈 등의 매칭(Matching)을 통해 공통성을 갖거나 또는 둘 사이의 구별할 수 있는 특징이 있어야한다. 이 과정에서 2개의 모듈 또는 프로시져에 공통된 특징이 많으면 유사성은 증가하나, 구별되는 특징 즉 단일성(Monotonicity)을 가지면 유사성이 감소하는 반비례의 관계를 갖는다.[1]

2개의 프로시져간 유사성(SIM: Similarity)은 아래와 같이 표현한다.

$$SIM(a, b) = F(f(a\cap b), f(a-b), f(b-a)) \quad (1)$$

단일성은 집합의 가중치에 영향을 받지 않으며 포함된 속성에 기초한다.

Arch는 단일성보다는 유사성을 중심으로 다음과 같은 알고리즘을 제시하였다.[4]

$$Sim(A,B) = \frac{(W(a\cap b) + K + Linked(S,B))}{(\eta + W(a\cap b) + (D\times(W(a-b) + W(b-a)))} \quad (2)$$

여기서 상호연결정도를 나타내는 상수 K, 프로시져의 독립성 정도를 나타내는 상수 D, 유사성을 나타내는 정규화 상수 η은 각기 특징의 연관성을 표현하였으나 현실적 적용에는 많은 어려움이 따른다. 즉 소스코드가 어떤 형태인가, 설계자의 기본적 의도에 따라 많은 차이가 있기 때문이다. 이러한 점을 고려해 본 논문에서는 Arch의 알고리즘을 다음과 같이 유사성 및 단일성을 명확히 구분하여 제안한다.

$$Sim(A,B) = \frac{W(a\cap b)}{(W(a\cap b) + (W(a-b) + W(b-a))} \quad (3)$$

구별되는 특징은 단일성(Monotonicity)으로 나타내기 위해

$$Mon(A,B) = \frac{W(a-b) + W(b-a)}{W(a\cap b) + (W(a-b) + W(b-a))} \quad (4)$$

명칭들 외에 정보은폐 사항들의 영향력 표현은 가중치로 나타나게되는데 대부분 Shannon의 정보이론에 특정표시의 확률로 나타냈으나, 본 논문에서는 소프트웨어에 사용된 변수 및 상수의 종류, 사용횟수 등을 고려해 적용한다.

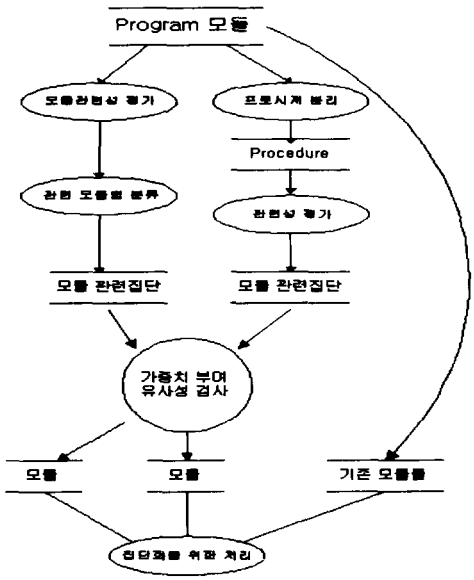
3. 소프트웨어 모듈의 집단화

3.1 프로그램 모듈의 집단화

리엔지니어링을 위해 소스코드로 구성된 모듈간에 관련성 있는 부분들을 집단화하는데 2가지 알고리즘을 제안한다.

- 모든 모듈로부터 프로시져들을 분리하여 프로시져간 유사성측정을 통해 집단화하는 방법
- 모듈간의 관련성을 휴리스틱한 측정방법을 이용해 집단화하는 방법

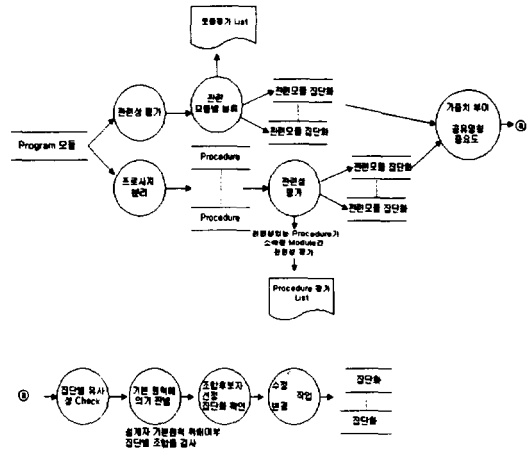
첫번째는 Arch가 제안한 알고리즘을 변경한 것으로 전반적으로 리엔지니어링 측면에서 매우 정확하고, 효과적인 결과를 유도할 수 있다. 그러나 효율성을 강조하다보면 소프트웨어의 많은 부분의 기본적인 변경·통폐합함으로 인해 소프트웨어 운영전체를 다시 확인해야하는 또다른 문제를 유발할 수 있어 소프트웨어 설계자의 기본적인 설계 의도와 다른 결과를 가져올 수 있다. 두번째는 전자의 방법을 해결하기 위해 설계자의 입장을 최대한 존중해 모듈들중 서로 관련성 있는 모듈들을 집단화하는 방식으로 설계자 의도의 긍정적 고려와 매우 빠르게 결과를 유도할 수 있다는 점에서 좋으나 효과적인 결과를 유도하느냐에는 문제가 있을 수 있다. 본 논문에서는 2가지 방법을 모두 적용하고 그 결과를 아래 기준에 근거해 가장 효율적인 방법을 선택한다.



(그림 1) 모듈간의 집단화 측정 과정
(Fig. 1) The process of measuring clustering of modules

- Arch 알고리즘의 변경·제안한 방법론이 설계자의 기본규칙에서 벗어나 있는가?
- 모듈 및 프로시져의 관련성을 통해 얼마나 정리될 수 있는가?
- 그 결과가 효과적이고, 유용한 가치를 지니고 있는가?

먼저 기존 모듈내에 소속된 프로시져들을 소속 모듈에서 해체해 프로시져간 정보공유라는 측면에서 공유변수등 프로시져간 공유명칭의 관련성을 중심으로 재분류 하고 집단화를 실행한다.



(그림 2) 집단화 처리 과정도
(Fig. 2) The flow chart of process of clustering modules

이 과정에서 프로시져가 소속된 모듈을 알수 있도록 <표 1, 2>의 자료가 필요하다. 이러한 자료를 통해 공유명칭들을 중심으로 프로시져들을 모듈 재집단화를 한다. 기존 프로그램 모듈을 구성하는 프로시져들을 그대로 두고 현 상태에서 모듈간의 관련성 즉 공유변수등 모듈간 공유명칭의 관련성을 중심으로 평가를 한 후 그 결과를 통해 관련성을 중심으로 기존 모듈들을 분류하고 집단화를 실행한다.

프로시져들을 분리하고 관련성을 중심으로 집단화하는 과정은 한번의 과정이 더 필요하게 된다. 즉 기존 모듈내에 있던 프로시져들을 해체한 후 관련성을 중심으로 사용횟수 및 중요도등을 중심으로 재모듈화를 실행하는 하나의 절차가 더 필요하게 된다. 이후 모듈단위의 관련성 평가를 통해 모듈들을 분류하고 재집단화하는 과정을 밟게된다.

<표 1>과 <표 2>는 모듈 및 프로시져간 집단화를 위한 자료로 사용할 수 있다. (그림 2)는 (그림 1)에서 나타나는 실제적인 과정을 포함해 각 집단별 유사성 측정을 나타냈는데 효율화 시킬 수 있는 집단화 방법

에 대한 고려사항 3가지는 아래와 같다.

1) 반복적으로 공유명칭들을 많이 가진 집단간의 통합여부
2) 집단내 각 프로시져의 위치가 적절한지를 파악후 조합 가능성 확인
3) 연산자의 조합을 통한 집단간의 통합 및 통합에 따른 보조집단의 생성여부

〈표 1〉 모듈간 공유명칭 관련성 평가 리스트 예
〈Table 1〉 The Evaluation list of the relatedness between modules in sharing names

모듈명(A)	반복명칭(A)	모듈명(B)	반복명칭(B)
kema	aks(15)	mem1	aks(10)
.	bks(12)	.	bks(9)

〈표 2〉 프로시져간 공유명칭의 관련성 리스트 예
〈Table 2〉 The list of the relatedness between procedures in sharing names.

모듈명 (A)	프로시져명	공유명칭 (A)	모듈명 (B)	프로시져명	공유명칭 (B)
kema	pgs	aks(15)	mem1	pme	aks(10)
"	"	bks(12)	"	"	bks(9)

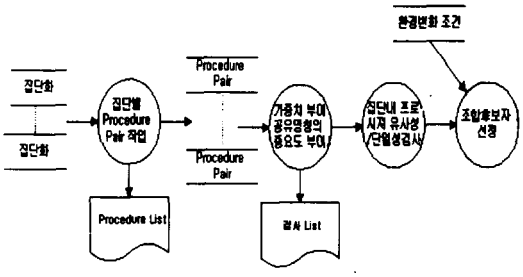
이를 통해 2가지 집단화된 모듈집단들을 기존 모듈과 비교하여 위 조건을 고려한 가장 바람직한 모듈집단을 선정하게 된다.

이 과정을 통해 통합 및 집단화를 위한 각 모듈 후보자를 선정하고 선정된 각 모듈들을 수정 및 변경을 통해 재집단화를 유도하게된다. 이 과정들은 각기 2개의 다른 집단화를 유도하게 된다. 즉 프로시져까지 완전히 해체해 관련성을 중심으로 집단화과정을 거치는 것과 기존 모듈을 인정하고 모듈간 관련성을 중심으로 집단화 과정을 거치는 과정, 그리고 본래의 모듈들 이 3가지를 평가하는 조건들을 통해 그중 가장 적절한 집단화 자료를 선정하게된다. 이 과정이후는 (그림 3)의 재집단화 과정으로 나타내었다.

3.2 프로그램 모듈의 재집단화

집단화과정에서 기본적인 평가기준을 검토하고 검토된 방법론 중 어떠한 방법론이 가장 좋은가를 판단해야 한다. 여기서 중요한 결정변수중 하나가 설계자의 의도이므로 필요시 설계자와 상의해야 한다. 집단내 프로시져를 풀어 정보공유측면에 집단화를 시도한다. 그 결과 현재의 모듈내 존재하는 프로시져보다 다른 모듈에서 사용되는 공유변수의 횟수 및 종류, 중요도에 따라 가중치 부여를 검토하고 이를 토대로 설계자가 집단간 유사성이라는 측면과 상위 및 하위 보조집단을 만들 필요가 있는지를 통해 대화형 형태로 재집단화를 유도한다. 가중치 부여는 반복횟수의 상대적 중량표시로 전체 반복횟수에 중위수 값을 구해서 이를 기준으로 상대적인 값을 적용한다. 아울러 공유명칭의 중요도는 일반적으로 평가대상 소프트웨어의 특성에 따라 달리 적용하게 되는데 본 논문에서는 프로시져의 호출, 외부상수 및 변수, 마크로명, 정적변수, 구조체명, typedef명, 기타 비지역명칭 순서대로 가중치를 부가한다.

이 과정의 결과를 통해 설계상 결함이라는 잘못된 모듈 및 프로시져의 수를 최소화할 수 있는 방법을 찾을 수 있다. 이후 본 논문에서 제시한 가중치를 고려해 프로시져간 정보공유 여부를 결정할 수 있는 유사성 및 단일성을 조사한다.



(그림 3) 재집단화 처리 과정
(Fig. 3) The process of Re-clustering

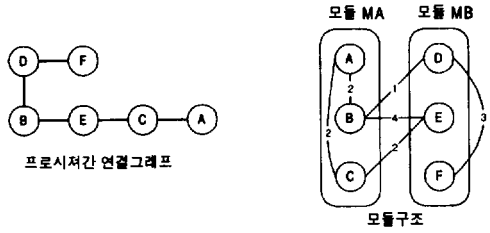
4. 실험 및 고찰

4.1 유사성의 실행

A, B, C 3개의 프로시져로 구성된 모듈 MA와 프로시져 D, E, F로 구성된 모듈 MB가 (그림 4)와 같이 프로시져간 호출관계로만 구성되었을 때 프로시져 한가지 특성만으로 제한하여 가중치를 무시하고, 설계가 잘못된점을 고려한다면 모든 프로시져들간의 유사성은 (그림 4)와 같은 연결형태를 갖는 집단으로 볼 수 있다.

이때 연결 유사성 기준을 0부터 조금씩 높여보면

유사성이 가장 낮은 B-D간의 연결이 끊어져 D-F와 B-E-C-A의 모듈로 (그림 6-1)과 같이 재구성된다. 같은 방법으로 모듈화 정도를 더욱 높이면 A-B간 연결과 C-E간 연결이 차례로 끊어져 (그림 6-2)와 같은 연결그래프로 D-F, B-E, A-C 3개 모듈로 동일한 방법을 모듈화를 높이면 A-B간, C-E간 연결이 차례로 끊어지어 재구성될 수 있다.



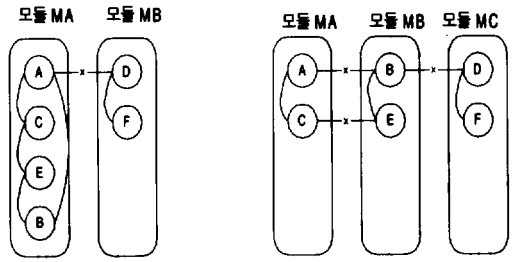
(그림 4) 프로시저간의 연결그래프 및 모듈구조
(Fig. 4) Connection Graph and Modules structure between procedures.

	A	B	C	D	E	F
A	1	2/9	1/3	0	0	0
B		1	0	1/10	4/9	0
C			1	0	1/4	0
D				1	0	3/4
E					1	0
F						1

→

3/4	D, F
4/9	B, E
1/3	A, C
1/4	C, E
2/9	A, B
1/10	B, D

(그림 5) 프로시저간 호출의 우선순위
(Fig. 5) Call priority of procedures structure



(그림 6) 모듈구조의 재구조화
(Fig. 6) Re-sturcturing of Module structure

4.2 프로그램 모듈의 집단화 실행

실험에 사용된 샘플 프로그램의 구성은 C언어로 작성되었으며 32개 모듈과 123개의 프로시저로 되어있다. 2가지 기본적인 집단화 방법의 실행을 위해 작성되었으며 32개 모듈과 123개의 프로시저로 되어있다. 모듈구조중 1단계 작업으로서 집단화를 위해 2가지 제안된 알고리즘을 적용하였다. 첫번째 알고리즘을 실행하여 가장 가까운 모듈을 순서대로 후보자 리스트를 만든다.

〈표 3〉 예비 집단별 후보리스트의 예
〈Table 3〉 The Examples of Candidates in preparatory group

기존모듈명	상대모듈명	프로시저명(A)	프로시저명(B)
kema	mem1	pgs	pme
kema	mem1	pgt	pmt

두번째 제안한 방법에 따라 가장 많은 공유명칭을 가진 프로시저들부터 하나이상의 정보공유명칭을 순서대로 후보집단별로 재모듈화를 실행한 후 관련성을 가진 모듈간에 예비 집단화를 실행한다.

이를 통해 다음과 같은 기존 모듈집단과 상기의 작업으로 구성된 후보집단간의 차이를 찾을 수 있다.

2개의 집단화 알고리즘을 적용해 만들어진 모듈집단들과 기존 집단화된 모듈집단간의 비교·검사를 통해서 어떠한 집단을 이용하는 것이 바람직한지를 판단하게 된다.

〈표 4〉 예비 집단별 후보리스트의 예
〈Table 4〉 The Examples of candidates in preparatory group

기존모듈명	프로시저명(A)	새모듈명	프로시저명(B)
kema	ags1	kema1	agg1
.	.	.	.

본래 모듈들은 7개의 그룹과 3개의 그룹 모듈들로 구성되었으며 각 그룹내에서 프로시저들의 유사성을 기준으로 재구성하여 선택한 모듈구조는 4개의 흡수된 모듈들과 2개의 분리된 모듈들로 인하여 30개의 모듈로 되었다. 또 본래 모듈들을 무시했을 때 3개의 흡수된 모듈들과 4개의 분리된 모듈들로 인하여 33개의 모듈로 되었다. 두가지 방법의 결과로 나온 모듈구조중

두 번째의 결과를 선택하여서 집단화를 실행한다. 이 과정에 가중치 부여조건은 본 논문에서 제시한 순위값 및 사용횟수를 그대로 고려하였다. 집단간의 유사성 측정결과를 통해 효율화를 시킬 수 있는 과정들을 필요로 하는데 여기서 고려해야할 사항은 아래와 같다.

- 모듈간에 동일한 기능을 수행하는 프로시져의 공유여부
- 모듈내 일부 프로시져만 다른 모듈내 프로시져와 설계상 정보 공유여부
- 설계상 잘못된 위치에 있는 프로시져를 가진 모듈
- 설계상 정보를 공유하는데 프로그래머의 주관적 습관 개입여부
- 단일 모듈로 수합이 필요한 모듈의 가능성
- 잘못된 전역변수의 사용여부
- 사용하지 않거나 거의 사용하지 않는 프로시져를 가진 모듈
- 기타 재조직과정이 필요한 모듈
- 자료구조는 같으나 동일한 기능의 수행여부

4.3 유사성 측정의 실행 결과

상기의 유사성 실행 방법을 이용해 프로시져간, 모듈간 유사성 측정을 실행한 후 변경 및 통폐합을 위한 평가를 한다. 이러한 집단화의 결과로 모듈구조는 33개의 모듈과 123개의 프로시져를 적용하여 중복되는 프로시져 2개, 사용하지 않는 프로시져 6개등 전체적으로 모듈 4개, 전역변수 등 전역명칭 14개의 삭제가 가능한 것으로 결론을 얻을수 있었다. 이 과정에서 외부변화는 고려하지 않았다. 결과적으로 재집단화된 프로그램 29개의 모듈과 115개의 프로시져를 가지게 되었으며, 중복 또는 미사용 프로시져를 갖지 않으며, 미사용의 전역변수를 갖지 않고, 각각의 모듈들은 비교적 높은 모듈성을 갖는 것으로 판별할 수 있다.

〈표 5〉 유사성 측정 실행결과
〈Table 5〉 The result of Similarity measurement

	본래갯수	중복갯수	미사용 갯수	변경 및 삭제	최종갯수
모듈수	33	4		4	29
프로시져 수	128	2	6	6	115
전역명칭의 수	·	·	·	14	·

5. 결 론

실험을 통해 집단화를 만들어 가는 과정에서 제일 중요한 것은 리엔지니어링 대상 소스프로그램의 특성임을 알 수 있었다. 이것은 설계자 및 프로그램 작성자에 의해 매우 큰 영향을 받는 것으로 보여진다. 정보공유라는 기본적 개념을 모듈의 관련성이라는 측면에서 본 논문에서 제안한 휴리스틱한 방법을 유사성 및 단일성 측정 알고리즘을 평가한 결과 높은 모듈성을 갖는 것을 알 수 있었다.

집단화 과정에서 각 모듈 및 프로시져에 공유되는 명칭의 종류, 사용 횟수등에 따라 집단화를 위한 평가기준은 각기 달라질 수밖에 없다는 것을 확인하게되었다. 결과적으로 집단화를 위한 평가기준의 정형화는 쉽지않은 것으로 보이며 추후 객체 및 다양한 종류의 프로그램등에 적용을 통해 각 프로그램 특성에 따른 집단의 정형화에 가능성을 연구할 필요가 있다.

참 고 문 헌

[1] Amos Tversky, "Features of Similarity", Psychological Review, Vol.84, No.4, July 1977.

[2] David L. Parnas, "On the Criteria To Be Used in Decomposing Systems into Modules", Tech. Report, Computer Science Department, Carnegie Mellon University, Aug. 1971.

[3] 송영재, 소프트웨어 엔지니어링, 1996, 홍릉출판사

[4] Robert W. Schwanke, "An Intelligent Tool for Re-engineering Software Modularity", Conference on Software Engineering, IEEE, pp.83-92, 1991.

[5] D.L. Parnas, "A Technique for the Specification of Software Modules with Examples", CACM, Vol.15, No. 5, pp.330-336, 1972.

[6] 변정우, 송영재, "소프트웨어 부품들의 재사용 품질평가를 위한 실용적 메트릭", 한국정보과학회 전문대학학술논문지, 제5권, 제1호, pp.81-88. Jun, 1997.

[7] 변정우, 송영재, "휴리스틱 측정방법을 이용한 소프트웨어 모듈평가에 관한 연구", 제9회 한국정보처리응용학회 춘계학술발표대회, 1998, 4. 6

[8] Jan Beck & David Eichmann, "Program and

Interface slicing for Reverse Engineering", IEEE, pp.509-518, 1993.

[9] Gerado Canfora, Aniello Cimitile, "A Logic Tools Production", IEEE TRANS SW Eng, Vol.18, No.12, DEC, 1992.

[10] Scott Burson, Gordon B. Kotik and Lawrence Z. Markosian, "A Program Transformation Approach to Automating Software", IEEE, pp.275-283, 1990.

[11] P. T. Breuter & K. Lano, "Creating Specification from code : Reverse-Engineering Tech nique", Software Mainenance : Research & Practice. Vol.3. pp.145-162, 1991.

[12] Harry M. Sneed & Agnes Kapos:, "A study on the Effect of Reengineering upon Software Maintainability", IEEE, SW Eng. pp.223-231, 1990.

변 정 우

1983년 경희대학교 전자공학과(학사)

1986년 한양대학교 산업대학원(석사)

1995년 경희대학교 대학원 전자계산공학과(박사과정)

1983년~1986년 경희대학교 전자계산소 근무

1987년~1995년 오산전문대학 전자계산과 조교수

1996년~현재 경희호텔경영전문대학 호텔경영과 부교수, 학생처장 겸 관광정보연구소장

관심분야 : 소프트웨어공학, 리엔지니어링, OOP, S/W 재사용

송 영 재

1969년 인하대학교 전기공학과(공학사)

1976년 일본 Keio Univ. 전산학과(공학석사)

1979년 명지대학교 대학원(공학박사)

1971년~1973년 일본Toyo Seiko연구원

1980년~1982년 공업진흥청 공업표준 심의위원

1982년~1983년 미국 Univ. of Mary land전산학과연구교수

1984년~1989년 전국대학전자계산소장, 협의회 총무이사, 부회장

1984년~1989년 경희대학교 전자계산소장

1989년~1990년 일본 Keio대학 전자계산과 객원교수

1976년~현재 경희대학교 전자계산공학과 교수

1993년~1995년 경희대학교 교무처장

1996년~현재 경희대학교 공과대학장

관심분야 : 소프트웨어엔지니어링, Oject Oriented Programming & System, CASE도구, S/W개발 도구론, S/W재사용