공학 연구논문집 제29권 2호 pp. 669~680, 울산대학교.
Journal of Engineeringl Research Vol. 29, No. 2, pp. 669~680, 1998
University of Ulsan.

재모듈화를 통한 소프트웨어 재사용의 지원*

이명재
컴퓨터 · 정보통신공학부

<요 약>

재사용 가능한 자산을 획득하는 가장 좋은 방법은 기존 소프트웨어 시스템으로부터 재사용 가능한 부품을 추출하는 것이다. 소프트웨어 재모듈화 기법은 시스템 내에 밀접히 연관되어 있는 요소들을 모아 시스템의 모듈성을 향상시키는 방법이다. 본 논문에서는 재모듈화 방법을 통해서 소프트웨어 재사용을 지원하는 방법을 제안한다. 재모듈화 방법은 소프트웨어 재사용의 네 가지 수준인 함수 재사용, 모듈/객체 재사용, 부시스템 재사용, 응용 시스템 재사용을 지원한다.

Supporting S/W Reuse through Remodularization Method

Myeong-Jae Yi
School of Computer Engineering and Information Technology

<Abstract>

The most promising approach for obtaining reusable assets is extracting reusable components from the existing software systems. Software remodularization is a method for enhancing system modularity by grouping closely related components in a system. In this paper, we present a method supporting software reuse through remodularization method. Remodularization method supports four levels of software reuse which are function reuse, module/object reuse, subsystem reuse, and application reuse.

---

* 이 논문은 1997학년도 울산대학교 학술연구조성비에 의하여 연구되었음.

1. 서 론

소프트웨어 재사용(reuse)은 '소프트웨어 위기'를 해결하는 것을 돕는 바람직한 접근법이다. 설계와 요구사항 문서 또는 코드의 일부분과 같은 소프트웨어 부품의 재사용은 새로운 소프트웨어 프로젝트의 품질과 생산성 모두를 향상시킬 수 있다.[1] 그러나, 불행하게도 많은 어려움들이 재사용이 산업계의 소프트웨어 생산 환경에서 광범위하게 사용되는 것을 힘들게 한다. 이러한 어려움의 대부분은 재사용 가능한 부품의 부족과 관련이 있다.

소프트웨어 재사용에 관한 격언 중에는 '당신이 무엇인가를 재사용하기 전에는, 반드시 재사용할 수 있는 무엇인가가 있어야 한다'라는 것이 있다[2]. 또한, Biggerstaff는 재사용에 대해서 '소프트웨어 재사용은 저축과 같다. 당신이 이자를 받으려면, 반드시 예금을 해야만 한다. 많이 저축하면 할수록, 더 많은 이자를 받을 수 있다'고 표현하였다[3]. 또한, 최근에는 Arnold와 Frakes도 재사용에 있어서 가장 긴박한 문제점은 재사용 가능한 자산을 찾아내는 것이라고 하였다[4].

재사용 가능한 자산을 획득하기 위한 방법으로는 세 가지가 있을 수 있다: 그들을 구입하거나, 특별히 재사용을 위해 개발하거나, 기존의 소프트웨어 시스템으로부터 재사용가능한 부품들을 추출하는 것이다. 이중 마지막 방법이 가장 유망한 방법이고, 또 이것만이 가까운 시일 내에 유용한 결과를 얻을 수 있는 방법일 것이다.

재모듈화 방법은 기존 소프트웨어 시스템의 외부적인 행위(기능적인 것과 의미론적인 것)에는 변경 없이, 동일한 추상화 수준에서 한 표현 형태를 다른 표현 형태로 변환하는 작업인 재구조화(Restructuring)와 마찬가지로 시스템의 기능을 유지하면서 시스템의 구조를 변형시키는 작업이다[5].

따라서, 원시 코드에 대해 재모듈화 작업을 성공적으로 수행하게 되면, 시스템의 전체 구조를 재사용하기 쉬운 구조, 즉 재사용가능한 부품들을 쉽게 추출할 수 있는 구조로 바꾸게 된다.

본 논문에서는 이러한 작업을 성공적으로 수행할 수 있는 재모듈화 방법을 소프트웨어 재사용에 적용하는 방법을 제시하고자 한다.

본 논문의 구성은 다음과 같다. 제 2 장에서는 소프트웨어 재사용의 여러 수준과 기존의 소프트웨어로부터 재사용가능한 부품들을 추출하는 방법, 그리고 여러 가지 재모듈화 방법들에 대해 기술한다. 제 3 장에서는 소프트웨어 재사용을 효과적으로 지원할 수 있는 재모듈화 방법을 설명하고 그 방법이 소프트웨어 재사용의 4가지 수준에 어떻게 적용될 수 있는지를 기술한다. 마지막으로 제 4 장에서는 결론을 맺고 향후 연구 과제에 대해서 기술한다.

2. S/W 재사용과 재모듈화 방법

2.1 S/W 재사용

대부분의 공학 분야에서의 설계 과정들은 부품의 재사용에 기초를 두고 있는데 반해서, 소프트웨어 시스템 설계는 대체로 모든 부품들이 개발되어지는 시스템을 위해 특별히 구현되어야 하는 것을 가정한다. 윈도우 시스템 라이브러리와 같은 라이브러리들을 제외하

면, 모든 소프트웨어 공학자들이 인지하고 있는 재사용 가능한 소프트웨어 부품들에 대한 공통적인 기반이 없다. 그러나, 이런 상황은 서서히 변화하고 있다. 사람들은 동일한 소프트웨어를 반복해서 다시 개발하는 것이 아니라 소프트웨어 자산들을 재사용할 필요성을 느끼고 있다.

소프트웨어 재사용은 그림 1과 같이 여러 다른 수준에서 고려될 수 있다:

(1) *응용 시스템 재사용* 어떤 응용 시스템 전체를 재사용할 수 있다. 이것은 하나의 시스템이 여러 개의 다른 플랫폼 상에서 수행될 수 있도록 하는 것이다. 여기에서의 주요한 문제점은 소프트웨어가 이식 가능하도록 보증하는 것이다.

(2) *부시스템 재사용* 응용 프로그램내의 주요한 부시스템들을 재사용할 수도 있다. 예를 들면, 컴파일러의 일부분으로 개발된 렉시칼 분석기나 파서를 정적 분석기 혹은 역공학 도구와 같은 소프트웨어 재공학 관련 도구에서 재사용할 수 있다.

(3) *모듈 혹은 객체의 재사용* 특정한 기능들의 모음을 나타내는 시스템의 부품들을 재사용할 수 있다. 예를 들면, 이진 나무를 구현한 Ada 패키지 혹은 C++ 객체들을 다른 응용 프로그램들에서 재사용할 수 있다.

(4) *함수 재사용* 수학적인 함수와 같이 하나의 기능을 구현한 소프트웨어 부품들을 재사용할 수 있다.

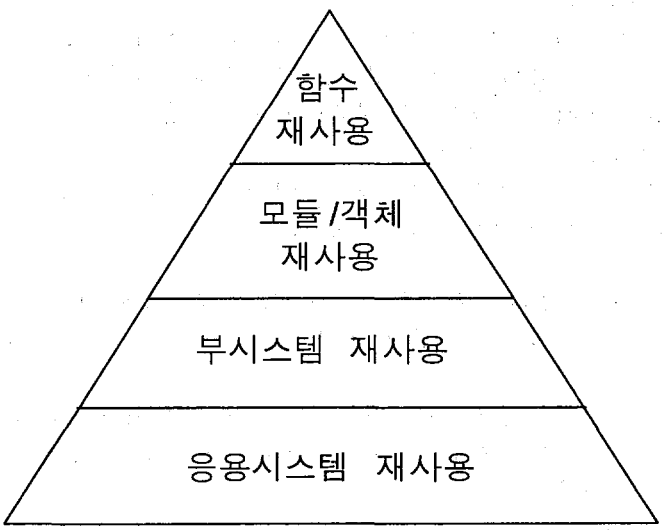

그림 1 소프트웨어 재사용의 여러 수준

소프트웨어 재사용이 소프트웨어 개발 작업에 보다 많이 이용되고 적용되려면, 이용가능한 재사용가능한 부품들이 많이 있고, 그러한 부품들을 쉽게 사용할 수 있도록 분류와 검색 시스템이 잘 구비되어야 한다. 재사용가능한 부품들을 많이 확보하기 위한 방법 중의 하나는 이미 있는 기존의 응용 소프트웨어들로부터 재사용가능한 부품들을 자동으로 추출하는 방법이다.

기존의 응용 소프트웨어들로부터 재사용가능한 부품들을 자동으로 추출하고 다른 소프트웨어의 개발에 활용하기 위한 과정은 그림 2에 나타낸 것과 같이 다음과 같은 다섯 단계를 거치게 될 것이다.[6]

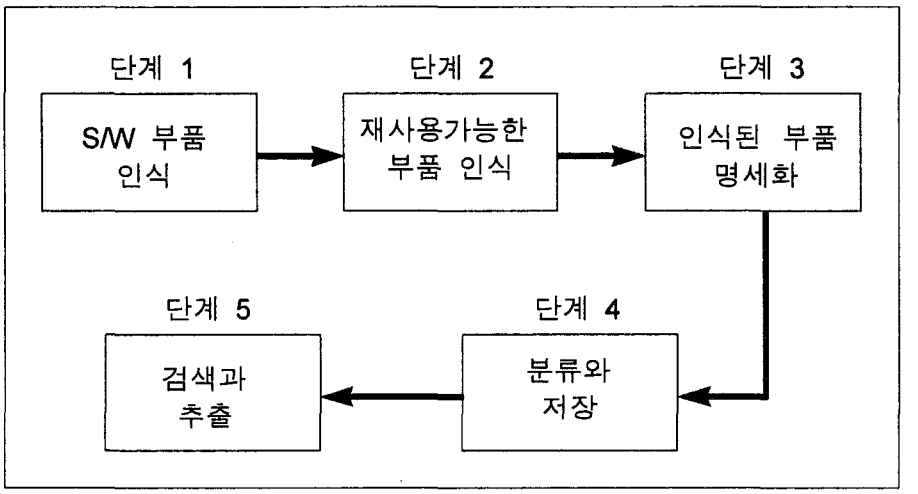

그림 2 기존 소프트웨어로부터 재사용가능한 자산을 획득하는 프로세스

단계 1: S/W 부품 인식

이 단계에서는 원시 코드에 대한 분석 작업을 수행하고 소프트웨어 부품들의 집합을 생성하는 단계이다. 이 집합들은 적절히 분할되거나, 합쳐지거나, 일반화되는 작업을 거쳐서 재사용가능한 부품들이 될 수 있는 후보들이다.

단계 2: 재사용가능한 부품 인식

이 단계에서는 전 단계에서 생성된 소프트웨어 부품 집합들에 대한 분석 작업을 수행함으로써 재사용가능한 부품들을 생성하는 작업이다.

단계 3: 인식된 부품 명세화

이 단계는 단계 2에서 생성된 재사용가능한 부품들에 대해 명세서를 작성하는 단계이다. 기능적 명세서와 인터페이스 명세서가 이 단계에서 작성되어야 한다.

단계 4: 분류와 저장

이 단계는 재사용가능한 부품들과 그에 관련된 명세서들을 분류하고 저장하는 단계이다. 이 단계의 목적은 정보저장소를 정의하고, 그곳에 생성된 재사용가능한 부품들을 채우는 것이다.

단계 5: 검색과 추출

이 단계는 정보저장소와 상호 작용할 수 있도록 프론트-엔드 사용자 인터페이스를 구축하는 단계이다. 이 단계의 주요한 목적은 예를 들면 정보저장소내를 쉽게 항해할 수 있도록 비쥬얼 인터페이스를 제공하는 것처럼, 사용자들이 가능하면 쉽게 재사용가능한 부품들을 찾을 수 있도록 하여주는 것이다.

이들 단계중 단계 1부터 3까지는 소프트웨어 역공학과 재공학 기술을 필요로 하는 단계들이고, 특히 단계 1의 작업은 소프트웨어 재모듈화 기술을 적용함으로써 많은 효과를 얻을 수 있다.

2.2 소프트웨어 재모듈화

재모듈화 방법은 기존 소프트웨어 시스템의 외부적인 행위(기능적인 것과 의미론적인 것)에는 변경 없이, 동일한 추상화 수준에서 한 표현 형태를 다른 표현 형태로 변환하는 작업인 재구조화(Restructuring)와 마찬가지로 시스템의 기능을 유지하면서 시스템의 구조를 변형시키는 작업이다[5].

재모듈화(Remodularization)의 정의는 다음과 같다[7].

재모듈화는 시스템을 구성하는 요소들의 특성과 이들 사이의 결합도(coupling) 등에 의해 서로 밀접히 관련되어 있는 요소들을 모으는 클러스터링 분석을 통해 시스템의 모듈 구조를 바꾸는 방법이다. □

따라서, 원시 코드에 대해 재모듈화 작업을 성공적으로 수행하게 되면, 시스템의 전체 구조를 재사용하기 쉬운 구조, 즉 재사용가능한 부품들을 쉽게 추출할 수 있는 구조로 바꾸게 된다.

지금까지 제안된 소프트웨어 재모듈화 작업과 관련된 방법들은 크게 함수 사의의 관계 분석을 중심으로 하는 방법과, 시스템에서 사용되는 자료를 중심으로 하는 방법, 또한 함수와 자료사이의 관계 분석을 중심으로 하는 방법으로 나누어 볼 수 있다.

(1) 함수 중심적 방법

함수 중심적 방법은 재모듈화 방법 중 가장 단순한 방법이라고 할 수 있는데, 이 방법의 기본 원칙은 모듈을 인식하기 위하여 하나의 루트를 설정한 후에 그 루트로부터 직·간접적으로 호출되는 모든 함수들을 인식하여 하나의 모듈로 설정하는 방법이다. Ccalls에서는 하나의 루트노드에 의해 지정되는 모듈을 모듈에 속하지 않는 다른 어떠한 함수들에 의해서도 호출되지 않는 함수들의 그룹으로 정의하였다, 그리고, 모듈을 인식하기 위하여 루트노드에 의해서 직·간접적으로 호출되는 모든 함수들을 인식하여 하나의 모듈로 설정하였다.[8] CIAS는 모듈을 인식하기 위하여 함수 사이의 단순한 호출 관계를 적용하여 컴파일 가능한 단위를 생성하는 것으로 모듈에 소속된 함수에서 사용하는 변수, 자료형(data type), 마크로 등을 모두 포함하고 있다.[9] Basili 등이 CARE 시스템에서 모듈을 독립적으로 재사용할 수 있도록 제안한 방법도 CIAS와 비슷한 방법으로 프로그램 단위들을 추출하고 있다.[10] 결국 이들 방법은 함수 사이의 단순한 호출 관계를 적용하고, 함수에서 사용되는 모든 자료를 포함함으로써 모듈을 인식하고 있는데, 이 방법은 시스템의 모든 구성 요소사이에 얽혀 있는 복잡한 관계를 이해하고 단순화하는 데에는 커다란 도움을 주지 못한다고 할 수 있다.

(2) 자료 중심적 방법

이 방법은 시스템에서 사용한 자료들의 사용에 대한 분석을 통하여, 밀접한 관련을 갖는 자료들을 묶음으로써 시스템의 모듈성(modularity)을 향상시키고자 하는 방법이다. 이 방법은 자료 객체를 생성하거나 추상 자료형의 생성 등에 사용되는 방법이다. Hevner와

Linger는 정규 수식(regular expression)을 이용해서 변수 사용에 대한 분석을 수행하고, 두 개의 자료가 동일한 분장에서 사용되는 빈도에 대한 분석을 통해 자료 객체를 구성하는 방법을 제시하였다.[11] Liu와 Wilde는 객체를 인식하는 방법중의 하나로 전역 변수에 기초한 객체 인식 방법을 제안하였다. 이 방법은 프로그램을 구성하는 함수와 전역 변수의 사용에 대한 분석 작업을 수행한 후에, 전역 변수와 이를 사용하는 함수들을 연결한 후 강결합 요소(strongly connected component)를 구성함으로써 객체를 구성하는 방법이다.[12] Tomic은 시스템에서 사용하는 자료를 사용하는 부분들을 추출함으로써 객체를 생성하는 방법을 사용하였다.[13] Canfora 등은 재사용가능한 추상 자료형(abstract data type) 을 추출하기 위하여 전역 변수와 그것을 사용하는 함수사이의 연결을 표시한 변수 참조 그래프(VRG)를 생성하고, VRG에서 내부 연결 부그래프를 추출하는 방법을 제시하였다.[14]

(3) 혼합적인 방법

이 방법은 함수 호출, 함수와 자료 사이의 관계, 그리고 자료형 등에 대한 분석을 혼합하여 사용한 방법들이다. Liu 와 Wilde는 객체를 인식하기 위한 또다른 방법으로 자료형에 기초한 방법을 제안하였다. 이 방법은 프로그램내의 모든 자료형에 대해 위상 순서(topological order)를 정의하고, 함수의 형식 매개변수와 리턴 값의 자료형에 대한 분석을 통하여 객체를 구성한다.[11] 한편, Livadas와 Roy는 전역 변수나 참조 호출 방법으로 전달되는 매개변수의 자료형과 이들을 수정하는 모든 함수들을 묶어 객체를 생성하는 수신자(receiver)에 기초한 객체 생성 방법을 제안하였다.[15]

Belady와 Evangelisti는 시스템내의 프로시쥬어들을 모듈로 클러스터링하기 위하여 used data binding을 사용하였다. used data binding은 두 모듈 p,q가 변수 x를 참조 또는 수정하기 위하여 사용한 것을 나타내는 것으로, 두 모듈 p,q 사이의 유사성을 나타낸다. 이들은 이 방법을 통하여 시스템을 계층을 갖지 않는 모듈로 나누었다.[16] Hutchens과 Basili는 control flow data binding(모듈 p가 제어를 가진 후에 모듈 q로의 제어 이동 가능성이 있고, p가 변수 x에 값을 지정하고, q가 x를 참조하는 경우)을 사용하여 프로그램내의 유사한 함수들을 클러스터링 하는 방법을 제시하였다. 또한 이들은 모듈들이 그룹화 되는 과정을 나타내기 위해 트리 형식의 역수형도(dendrogram)를 생성함으로써 모듈들의 계층을 생성하였다.[17] Schwanke는 설계 결합도 개념을 사용하고, 프로시쥬어에 있는 모든 비지역적 이름(non-local name)을 프로시쥬어의 특징(feature)으로 정의하고, 각 특징 f 마다 특징 f가 사용된 횟수에 반비례하는 가중치를 두었다. 그는 두 프로시쥬어 a와 b의 유사성을 측정하기 위해 a와 b에서 공유되는 특징뿐만 아니라, a와 b에 각각 고유한 특징들을 고려하였고, a와 b 사이의 직접적인 호출 관계의 종속성도 고려하였다.[18] Patel 등은 프로그램에서 사용한 변수의 자료형을 가지고 모듈들의 응집도(cohesion)를 증가시키는 것에 의해 모듈들의 그룹화를 수행하는 방법을 제안하였다.[19] Muhler와 Uhl은 낮은 결합도와 높은 응집도를 갖도록 하면서, 계층화된 그래프인 (K,2)-partite 그래프를 생성하는 방법을 제안하였고[20], Choi와 Scacchi는 모듈간의 결합도(coupling)를 최소화하고, 자원 흐름도(Resource Flow Diagram)에서 변경 거리(alteration distance)를 최소화하는 방법에 의해 모듈들을 부시스템에 할당하는 알고리즘을 제안하였다.[21]

재모듈화 작업을 효과적으로 수행하기 위해서는 다음과 같은 사항을 만족하여야 한다.

첫째, 두 함수 사이의 간접적인 호출에 대한 특성을 고려하여야 한다. 예를 들어 함수 A

가 B를 호출, B는 C를 호출하는 경우에 함수 A의 변경이 B에는 물론 C에도 영향을 미치게 되고 그 반대의 경우(C를 수정하였을 경우)도 성립하므로 직접적인 호출뿐만 아니라 간접적인 호출에 대한 특성을 고려해야 한다. 둘째, 함수와 함수 사이의 유사성을 측정하는데 있어서 함수 내부에서의 자료 흐름은 크게 중요하지 않으므로, 전역 변수에 대한 특성만을 고려하는 것이 바람직하다. 셋째, 상위 계층의 자료형을 가진 변수에 대한 사용 권한은 하위 계층의 자료형에 대한 사용 권한까지 함께 갖게 되므로, 자료형의 계층에 대한 특성을 고려하여야 하고, 마지막으로 두 함수사이의 유사성을 고려하기 위해서는 공통의 특성뿐만 아니라 서로간에 다른 특성에 대한 것도 함께 고려되어야 할 것이다.

그러나, 기존의 모든 방법들은 이 모든 사항을 만족하고 있지는 않고 있다. 이에 본 논문에서는 앞의 요구조건들을 모두 만족하며, 인식되는 그룹들 간의 계층을 명확히 하기 위하여 각 단계별로 그룹 인식 작업을 수행하는 재모듈화 방법을 통해 이 방법이 소프트웨어 재사용에 어떻게 적용될 수 있는지를 논한다.

3. 재모듈화 방법을 이용한 재사용의 지원

3.1 재모듈화 방법

본 논문에서 사용하는 재모듈화 방법은 그림 3에 나타낸 것과 같이 크게 5가지의 단계로 구성되어 있다. 단계 1은 초기화 단계로 클러스터링 인식 작업에 사용되는 프로그램에서 잘못 사용되었던 모순 사항들을 제거하고, 프로그램을 단순화시키는 단계이고, 단계 2는 부시스템을 이루는 기본 단위인 함수들을 서로 밀접히 연관된 함수들로 묶는 함수 클러스터링 작업을 수행한다. 단계 3은 프로그램에서 사용되었던 자료들 중에서 밀접히 연관

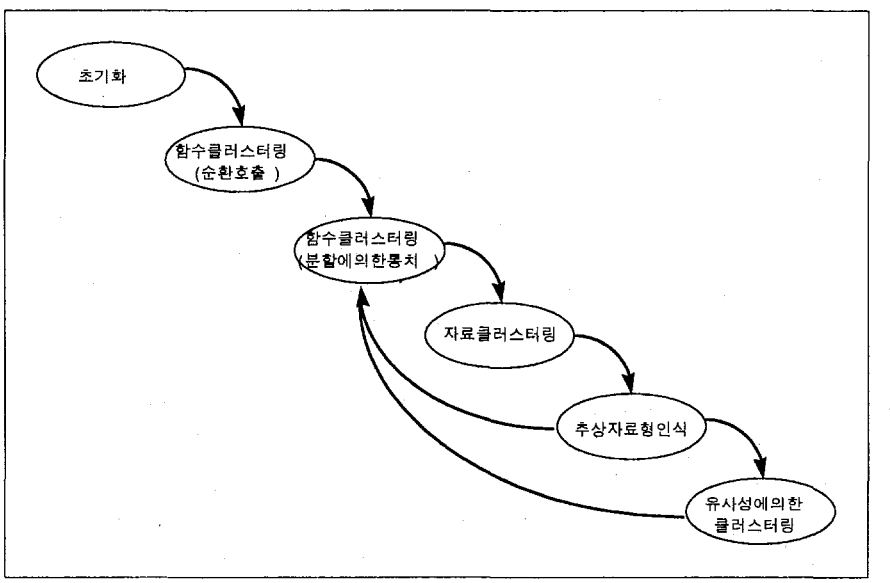

그림 3 재모듈화 방법

된 자료들을 하나로 묶는 자료 클러스터링 작업을 수행하며, 단계 4에서는 추상 자료형을 인식하는 단계, 마지막으로 단계 5에서는 함수와 자료, 자료형 사이의 연관 관계 분석을 통하여 유사성(similarity)을 측정하고, 유사한 함수들과 자료, 자료형 들을 클러스터링 하여 부시스템으로 생성하는 단계이다.

(1) 초기화

재모듈화 작업의 첫 단계는 인식 작업을 정확히 수행하기 위해 원래의 시스템에 내재해 있던 모순(anomaly)을 제거하는 것이다. 이러한 모순은 시스템의 수행 기능에 영향을 미치지는 않지만, 시스템을 이해하는데 어려움을 더욱 배가시키는 것들이다. 변수들 사이의 중복 정의(정의된 변수를 사용하는 것 없이 다시 재정의하는 경우), 정의되지 않은 변수의 사용 등과 같은 자료 흐름에 대한 모순이외에 본 논문에서 제안하는 방법에 영향을 미치는 다음과 같은 모순을 제거한다. 첫째로 프로그램에서 한번도 호출되지 않는 함수를 제거한다. 또한 이러한 함수에서만 호출되는 함수들을 재귀적으로 제거한다. 다음은 정의만 되고 한번도 사용되지 않는 변수들을 제거하고, 하나의 함수에서만 사용되는 전역 변수는 그 함수의 지역 변수로 정의한다.

(2) 함수 클러스터링

두 번째 단계는 시스템 개발 시에 적용하는 구조적 기법의 분할에 의한 통치 원칙을 적용시켜 관련된 함수를 클러스터링 하는 작업이다. 함수 클러스터링을 위해서 우선적으로 순환적인 호출을 형성하는 함수들을 하나의 클러스터로. 묶는다. 시스템 내에서 순환적 호출을 형성하는 함수들은 그들 사이에 밀접한 관련을 맺게 되는 것이 일반적이므로 이들을 하나의 클러스터로 형성한다. 이 작업은 호출자와 피호출자의 관계를 단순화하기 위해 수행한다.

다음은 많은 사람들이 프로그램 개발 시에 적용하는 분할에 의한 통치 원리를 적용하는 것이다. 프로그래머들이 함수를 만드는 이유는 크게 어떤 특정한 일을 수행하는 부분이 여러 곳에서 반복적으로 수행될 것이므로 함수를 만드는 일과, 하나의 함수의 크기가 너무 커지지 않도록 하기 위해 특정한 일을 수행하는 일련의 과정들을 묶어서 하나의 함수로 생성(분할에 의한 통치)하는 두 가지를 들 수 있다. 그런데 많은 프로그래머들이 첫 번째 이유보다는 두 번째 이유에 의해 함수를 생성하는 경우가 더 많은데, 이러한 이유로 생성된 함수는 대체로 하나의 호출자만을 갖게 된다. 이러한 함수들은 어떤 함수의 복잡도를 감소시키기 위해 함수의 기능을 분할하여 새로운 함수를 생성한 것이므로 이들 함수를 하나의 클러스터로 형성하는 것이 타당할 것이다. 단, 어떤 함수가 단지 하나의 호출 함수를 갖는다고 해서 무조건 그 함수를 호출 함수와 같은 클러스터로 생성하는 것은 위험하다. 왜냐하면 그 함수가 특정한 전역 변수에 대한 작업을 수행하는 함수라면 그 함수를 전역 변수와 연관지어 클러스터로 형성하는 것이 바람직하기 때문이다.

따라서 하나의 호출자를 갖는 함수가 사용하는 전역 변수의 집합이 호출자가 사용하는 전역 변수의 집합에 포함될 때에만 호출자와 같은 클러스터로 생성한다. 이 원칙은 반복적으로 적용되어, 새로 생성된 함수 클러스터를 하나의 호출자로 갖는 함수에 대해서도 함수 클러스터링 작업을 수행한다.

(3) 자료 클러스터링

이 단계는 시스템에서 사용되는 전역 변수들을 단순화하는 작업으로, 전역 변수의 사용이 동일한 함수 클러스터 내에서 이루어지는 전역 변수들을 하나의 자료 클러스터로 형성한다. 예를 들어 변수 v1과 v2 모두 함수 f1, f2, f3에서만 사용된다면, 두 변수 v1과 v2 사이에는 밀접한 연관을 맺고 있다고 유추할 수 있으므로 두 변수를 묶어 하나의 자료 클러스터로 생성한다. 이 작업은 작업의 효율성을 위해서 전역 변수의 사용 빈도가 낮은 전역 변수부터 순서대로 작업을 수행한다. 이 단계까지의 작업을 수행하게 되면 시스템의 가장 기본 요소인 함수와 전역 변수들이 단순화되어 다음 단계에서 사용할 전역 변수와 함수, 그리고 자료형의 관계 분석에 많은 도움을 줄 수 있다.

(4) 추상자료형 인식

이 단계는 함수와 전역 변수 사이의 관계를 분석하여 부시스템을 형성하는 단계이다. 이를 위해 추상 자료형의 추출을 위해 사용되었던 방법을 우선적으로 적용한다. 차이점은 전 단계까지 수행되었던 함수와 자료 클러스터가 함수들과 전역 변수들을 일부 대신하고 있는 점이다. 이 방법은 함수(함수 클러스터)들과 자료(자료 클러스터)들 사이의 관련성을 분석하기 위하여 이들 사이의 관계를 표시한 전역 변수 사용 그래프를 생성하고, 이 그래프에서 강결합 부그래프를 추출하는 것이다. 밀접히 관련된 함수와 전역 변수를 추출하기 위하여 모든 함수에 대하여, 함수가 사용하는 모든 전역 변수와 그 전역 변수들을 사용하는 모든 함수들을 연결하여 생성된 부그래프(sub-graph)의 내적 결합성(internal connectivity)을 정의하고, 어떤 함수의 내적 결합도와 합성된 요소들 간의 내적 결합도의 합과의 차이에 의해 어떤 함수를 중심으로 클러스터링 작업을 수행할 것인지를 결정한다.

모든 함수에 대하여 이 차이를 계산하여 이 값이 0.5 이상인 함수를 중심으로 함수와 전역 변수 클러스터링을 수행한다.

(5) 유사성에 의한 클러스터링

이 단계는 프로그램의 모든 구성요소를 고려하여 재모듈화 작업을 수행하는 단계이다. 이 단계에서는 지금까지 고려하지 않았던 자료형에 대한 것도 포함시켜서 함수 사이의 유사성을 구하여 가장 높은 유사성을 갖는 것들을 합병시켜 가는 과정을 수행한다.

두 함수사이의 유사성은 두 함수 A,B 사이에서 공통적으로 사용되는 특성 — 두 함수에서 공통으로 사용되는 전역 변수와 자동 변수, 형식 매개변수의 자료형 또는 리턴하는 자료형 중에서 공통적으로 사용되는 것, 두 함수가 공통적으로 호출하는 함수 또는 두 함수를 공통적으로 호출하는 함수 — 과, 각각의 함수에서만 사용되는 특성, 그리고 두 함수 사이의 호출 종속성, 그리고 유사성 수식이 0부터 1사이의 값을 갖도록 하기 위한 정규화 요소(normalizing factor)인 상수로 구성된다. 재모듈화 방법은 현재까지 인식된 함수 클러스터와 함수 클러스터에 포함되지 않은 각각의 함수들을 그룹으로 하여 두 개의 가장 유사한 그룹을 인식하여, 두 개의 그룹을 하나의 그룹으로 합친다.

3.2 재사용에의 적용 방법

본 절에서는 앞에서 기술한 재모듈화 방법이 소프트웨어 재사용의 여러 수준을 지원하는 방법에 대해 기술한다.

(1) 함수 재사용에의 적용

함수 재사용은 하나의 기능을 수행하는 함수를 재사용하는 것을 의미하므로 재모듈화 과정에 의해 새롭게 재사용가능한 하나의 함수를 생성하지는 않게 된다. 그러나 재모듈화 과정 초기 단계에서의 기존 소프트웨어에 대한 분석 과정을 통해 각 함수들에 대한 재사용에 대한 기준을 만족하는 지에 대한 정보를 축적할 수 있다. 즉, 그림 2에서의 단계 2인 재사용가능한 부품에 대한 인식 작업을 지원할 수 있다.

(2) 모듈 혹은 객체의 재사용에의 적용

재모듈화 방법은 이 수준의 재사용에 대해서 많은 지원을 하게된다. 재모듈화 방법의 단계 4인 추상 자료형 인식 단계까지 생성된 모든 클러스터들이 재사용가능한 부품에 대한 후보들이 될 수 있다. 특히 단계 4에서의 작업을 통해 관련된 함수, 자료, 자료형을 클러스터링할 수 있으므로 객체에 대한 생성 작업까지도 지원할 수 있다.

(3) 부시스템 재사용에의 적용

앞절에서 기술한 재모듈화 방법은 단계 1부터 단계 5까지 작업을 순환적으로 수행하면서 각 반복마다 클러스터링된 모듈들을 생성하게 된다. 또한 이렇게 생성되는 모듈들에 대한 수형도(dendrogram)를 통해 각각이 재사용가능한 부시스템의 후보들이 될 수 있다. 이렇게 생성되는 부시스템들에 대해 적절한 재사용 품질 기준을 적용함으로써 부시스템 전체에 대한 재사용을 지원하게 된다.

(4) 응용 시스템 재사용에의 적용

응용 시스템 재사용에서 가장 핵심이 되는 것은 기존 응용 시스템이 다른 플랫폼에 얼마나 잘 이식될 수 있나에 대한 응용 시스템 이식성(portability)에 관한 문제이다. 응용 시스템 이식성을 지원해주기 위해서는 응용 시스템 중에서 특정한 하드웨어나 운영체제 혹은 소프트웨어에 종속적인 부분들을 다른 부분들과 분리하여 이들에 대한 이식성 인터페이스를 개발하는 것이 좋은 방법이다. 앞절에서 기술한 재모듈화 방법은 특정한 하드웨어나 운영체제 혹은 소프트웨어에 종속적인 부분들을 부시스템으로 인식하게 된다. 그러므로 이렇게 인식된 부시스템들에 대해 이식성 인터페이스를 개발해주고, 나중에 다른 환경에 이전할 때 이식성 인터페이스는 그대로 유지하면서 그 내부만 새로운 환경에 맞게 변경시켜주면 응용 시스템 재사용에 대해 많은 성과를 얻게될 것이다.

4. 결 론

본 논문에서는 소프트웨어 재모듈화 방법을 재사용에 적용하기 위해 기존 재모듈화 방법에 대한 비교 평가를 하였고, 이를 통해 소프트웨어 재사용을 지원하기 위한 재모듈화 방법을 효과적으로 수행하기 위한 기준과 방법에 대해 기술하였다. 또한 기존의 소프트웨어에서 재사용가능한 소프트웨어 부품을 추출하는 과정과 재사용에 대한 여러 수준을 제시하고, 재모듈화 방법이 재사용의 여러 수준을 어떻게 지원하는 가에 대해서 기술하였다.

본 논문에서 제안한 재듈화를 통한 재사용의 지원 방법을 통해 소프트웨어 재사용 기법의 적용성 확장과 재사용 가능한 부품의 자산 증대의 기대 효과를 가져올 것으로 기대된다. 또한 부수적으로 재모듈화 작업을 통해 레거시 시스템의 이해가능성, 판독성, 유지보수성의 향상을 가져올 것으로 생각한다. 그리고 재모듈화 방법의 효과적인 적용을 통해 기존의 기능 중심 언어를 객체 지향 언어로 변환하는 작업을 수행할 수 있을 것이다.

향후 연구 계획으로는 재사용가능한 소프트웨어 부품에 대한 품질 기준을 설정하고 이에 따라 본문에서 기술한 기존 소프트웨어로부터 재사용가능한 부품의 확립 과정 모든 단계를 지원하는 프레임웍을 개발하는 것과 재모듈화 작업에서 사용되는 유사성에 대한 공식을 더욱 정제화하는 작업이 필요하리라 생각된다.

참 고 문 헌

[1] P. Freeman, Tutorial on Software Reusability, IEEE Computer Society Press, New York, 1987.

[2] W. Tracz, "Software Reuse Maxims," ACM SIGSOFT, Software Engineering Notes, 13(4), pp.28-31, 1988.

[3] T.J. Biggerstaff, "Design Recovery for Maintenance and Reuse," IEEE Computer, 22(7), pp.36-49, 1989.

[4] R.S. Arnold and W.B. Frakes, "Software Reuse and Reengineering," CASE Trends, 4(2), pp.44-48, 1992.

[5] E.J. Chicofsky and J.H. Cross, "Reverse Engineering and Design Recovery: A Taxonomy," IEEE Software, pp.13-17, Jan. 1990.

[6] R.S. Arnold, Software Reengineering, IEEE Computer Society, 1993.

[7] G.Canfora, A.Cimitile, M.Munro, "RE2: Reverse Engineering and Reuse Re-engineering," Journal of Software maintenance, 1992.

[8] D.R.Kuhn, "A Source Code Analyzer for Maintenance," Proc. of IEEE Conf. on Software Maintenance, pp.176-180, Nov. 1987.

[9] Y.Chen, M.Y.Nishimoto, C.V.Ramamoorthy, "The C Information Abstraction System," IEEE Trans. on Software Engineering, Vol. 16, No.3, pp.325-334, March 1990.

[10] S.K.Abd-El-Hafiz, V.R.Basili, G.Caldiera, "Towards Automated Support for Extraction of Reusable Components, " Proc. of IEEE Conf. on Software Maintenance, pp.212-219, Oct. 1991.

[11] A.R.Hevner, R.C.Linger, "A Method for Data Re-engineering in Structured Programs," Proc. of 22nd Hawaii Int'l Conf. on System Sciences, pp.1025-1034, 1989.

[12] S.Liu, N.Wilde, "Identifying Objects in a Conventional Procedural Language: An Example of Data Design Recovery, " Proc. of IEEE Conf. on Software Maintenance, pp.266-271, Nov. 1990.

[13] M.Tomic, "A Possible Approach to Object-Oriented Reengineering of Cobol Programs," ACM SIGSOFT SE Notes, Vol.19, No.2, pp.29-34, April, 1994.

[14] G.Canfora, A.Cimitile, M.Munro, C.J.Taylor, "Extracting Abstract Data types from C Programs: A CASE Study," Proc. of IEEE Conf. on Software Maintenance, pp.200-209, Sept. 1993.

[15] P.E.Livadas, P.K.Roy, "Program Dependence Analysis," Proc. of IEEE Conf. on Software Maintenance, pp.356-365, Nov. 1992.

[16] L.A.Belady, C.J.Evangelisti, "System Partitioning and its Measure," Journal of Systems and Software, Vol.2, No.1, pp.23-29, Feb. 1982.

[17] D.H.Hutchens, V.R.Basili, "System Structure Analysis: Clustering with Data Bindings," IEEE Trans. on Software Engineering, Vol.11, No.8, pp.749-757, Aug. 1985.

[18] R.W.Schwanke, "An Intelligent Tool for Re-engineering Software Modularity," Proc. of 13th Int'l Conf. on Software Engineering, pp.83-92, 1991.

[19] S.Patel, W.Chu, R.Baxter, "A Measure of Composite Module Cohesion," Proc. of 14th Int'l Conf. on Software Engineering, pp.38-48, 1992.

[20] H.A.Muller, J.S.Uhl, "Composing Subsystem Structures Using (K,2)-Partite Graphs," Proc. of IEEE Conf. on Software Maintenance, pp. 12-19, Nov. 1990.

[21] S.C. Choi, W.Scacchi, "Extracting and Restructuring the Design of Large Systems," IEEE Software, pp.66-71, Jan. 1990.