韓國OA學會論文誌
第1卷 第1號, 1996. 8

論文 96-1-1-3-3

역공학 기법을 적용한 설계정보의 회복에 관한 연구

서 민 호*

A Study on Recovery of Design Information Using A Reverse Engineering

Min Ho Seo*

요 약

소프트웨어의 전체 비용 중에서 유지보수 비용이 절반 이상을 차지하고 있기 때문에, 이 비용을 줄이기 위한 도구가 필요하다. 유지보수를 위한 공학적인 기법들 중에서 역공학은 원시 코드로부터 설계 정보를 추출하고 원시 코드를 다른 형태 혹은, 좀 더 추상화된 형태로 표현하는 과정이다.

본 논문에서는 역공학 기법을 적용하여, 기존 COBOL 프로그램으로부터 프로그램들 간의 호출 관계, 프로그램 내의 논리 구조와 프로그램의 자료 흐름에 대한 정보를 추출하는 방법에 대해서 제안하고 회복된 설계 정보로부터 프로그램 구조도와 모듈 구조도를 생성하는 방법에 대해서 제시한다.

원시 코드로부터 정확하게 생성된 구조도는 유지보수자가 프로그램을 전체적으로 혹은, 구체적으로 이해하도록 지원하는 중요한 정보가 될 것이다. 또한, 유지보수자가 프로그램을 재분석할 때 혹은 다른 유지보수자들이 그 프로그램을 분석할 때, 명확하게 수정된 구조도는 유지보수자들에게 프로그램을 더욱 쉽게 이해하도록 지원하게 될 것이다.

ABSTRACT

The maintenance cost accounts for over a half of all software costs. Maintenance tools can be used to reduce it. All other maintenance techniques, reverse engineering is a process of analyzing source code to extract design information and to create representation of it in another form or at the higher level of abstraction.

In this thesis, we propose a method to extract the call relationships among programs, the logic structure in program and the data flow of programs from COBOL source programs using reverse engineering. We also present a method to generate the structure chart of programs and modules.

The structure chart generated from source code provides very important information to understand programs in details. The structure chart modified will be more helpful the maintainer to understand programs when he analyzes them later or others analyze them.

* 광주여자전문대학 사무자동화과

Ⅰ. 서 론

1.1 연구 동기

기존 소프트웨어는 시간이 지남에 따라 많은 변경이 이루어져 복잡하게 되었고 또한, 새로운 소프트웨어는 환경의 영향을 받아 규모가 커지게 되었기 때문에, 1970년대부터 소프트웨어 유지보수의 비용이 지속적으로 증가하여 소프트웨어의 전체 비용 중에서 개발 비용보다 유지보수 비용의 비중이 높아지게 되었다[7]. 이러한 소프트웨어의 복잡성과 대규모성은 유지보수의 비용 중에서도 소프트웨어를 이해하는데 드는 비용을 가장 많이 차지하게 되는 원인이 되었다.

유지보수 작업을 수행하기 위해서, 유지보수자는 먼저 소프트웨어를 이해하여야 한다. 그러나, 다음과 같은 문제점들 때문에, 유지보수자가 소프트웨어를 이해하는데 많은 어려움을 겪고 있다. 소프트웨어는 일반적으로 프로그램, 문서와 기타 관련된 자료들로 구성되어 있다. 기존 소프트웨어를 이해하려면, 먼저 해당 문서를 참조하여 이해하고 프로그램을 분석하는 작업이 이루어져야 한다. 그러나, 해당 문서가 없거나 있다하더라도 프로그램과 일치하지 않는다면, 원시 코드만이 기존 소프트웨어를 이해하는데 가장 중요한 자료가 될 것이다. 그러나, 일반적으로 응용 소프트웨어들은 많은 프로그램들로 구성되어 있기 때문에, 원시 코드를 일일이 따라 가면서 전체 프로그램의 구조를 파악하는 일은 쉽지 않다. 그리고 프로그램 내의 복잡한 논리 구조와 입출력 자료를 수작업으로 추적하는 일도 엄청난 노력을 필요로 하는 일이다. 그러므로, 소프트웨어 이해에 드는 노력을 줄이기 위해서는 전체 프로그램의 구조, 프로그램 내의 논리 구조와 프로그램의 자료 흐름을 파악할 수 있는 도구가 필요하다.

1.2 연구 범위 및 목적

소프트웨어 생명주기(software life cycle)는 일반적으로 계획(plan), 개발(development)과 유지보수(maintenance) 단계로 이루어진다. 1970년대 이후로 유지보수에 드는 노력이 개발에 드는 노력보다 월등히 많아지게 되자 유지보수를 지원하기 위한 공학적인 기법들이 소개되었다[7][8]. 역공학, 재구성과 재공학은 이 기법들을 대표하는 것들이며 특히, 역공학은 원시 코드로부터 설계 정보를 회복하는 것이다.

역공학 기법을 적용하여 원시 코드로부터 회복되는 설계 정보에는 논리(logic)와 자료(data)에 대한 정보가 있다. 논리 정보는 프로그램의 기능, 프로그램들 간의 상호 관계와 프로그램 내의 절차를 표현한 것이고 자료 정보는 자료, 자료 참조와 자료 구조를 표현한 것이다. 본 논문에서는 논리와 자료에 대한 회복 정보로서 원시 코드로부터 전체 프로그램의 구조, 프로그램 내의 논리 구조와 프로그램의 자료 흐름을 표현하는 방법에 대해서 제안한다.

본 논문의 목적은 첫째, 원시 코드(COBOL로 작성된 프로그램)로부터 논리 정보인 전체 프로그램의 구조와 프로그램 내의 논리 구조를 파악하고 자료 정보인 프로그램의 입출력 정보를 추출하는 것이다. 이 정보들은 저장소에 적재된다. 둘째, 이 저장소의 설계 정보로부터 프로그램의 구조, 절차와 자료 흐름을 그림으로 표현하는 구조도 생성기를 설계하는 것이다. 셋째로 생성된 구조도를 의미가 있는 구조도로 수정하거나 혹은 자세히 볼 수 있도록 편집하는 구조도 편집기를 구축하는 것이다. 즉, 프로그램을 쉽게 이해하도록 지원하기 위하여 원시 코드로부터 구조도를 생성하는 시스템을 구축하는 것이다.

1.3 기대 효과

본 논문의 연구 결과로 나타날 수 있는 기대 효과로는 다음과 같은 것들이 있다.

(1) 유지보수를 위한 자동화 도구를 제공하므로 소프트웨어의 설계 정보를 자동으로 수집 할 수 있다.
(2) 프로그램의 구조와 입출력 자료를 시각적으로 표현하므로 유지보수자가 기존 프로그램을 쉽게 이해할 수 있다.
(3) 프로그램의 구조에 대한 정보를 저장소에서 체계적으로 관리하므로 재분석 작업에서 이 정보를 효과적으로 재 사용할 수 있다.

Ⅱ. 소프트웨어 유지보수와 역공학

2.1 소프트웨어 유지보수

소프트웨어 생명주기는 계획 단계, 개발 단계(분석, 설계와 구현 단계)와 개발 이후의 단계(유지보수 단계)로 구분할 수 있다. 일반적으로 자원과 비용을 소모하는 입장에서, 소프트웨어 개발을 '빙산의 일각'이라 불릴 만큼 소프트웨어 유지보수는 소프트웨어 공학에서 상당히 어려운 분야로 알려져 왔다. 이는 소프트웨어의 자원과 비용의 대부분을 유지보수단계에서 소비하고 있기 때문이다.

2.1.1 유지보수의 정의 및 과정

소프트웨어 유지보수는 개발 단계에서 만들어진 소프트웨어를 사용자에게 인도한 다음에 이루어지는 활동으로서 완성된 소프트웨어를 수정하고 보완하는 활동이다. 그리고, 유지보수 단계에서 수행하는 활동에 따라 교정, 적응과 완전 유지보수로 분류한다[6][15].

(1) 교정 유지보수(corrective maintenance)는 소프트웨어 개발 과정에서 발생하는 오류를 시험 단계에서 모두 발견하여 정정할 수 없기 때문에 수행하는 활동이다.
(2) 적응 유지보수(adaptive maintenance)는 컴퓨터 환경의 변화를 기존 소프트웨어에 반영하기 위하여 수행하는 활동이다.
(3) 완전 유지보수(perfective maintenance)는 기존 소프트웨어가 정상적으로 작동하지만, 현재 기능의 수정이나 새로운 기능의 추가 그리고 전반적인 성능 개선 등을 사용자로부터 요구받았을 때 수행하는 활동이다.

이러한 유지보수 활동들은 (1)유지보수하고자 하는 문제를 인지하는 단계, (2)그 문제와 관련된 프로그램 및 자료를 파악하고 이해하는 단계, (3)코드를 변경하는 단계, (4)변경된 코드를 시험하여 원하는 결과인가를 확인하고 그 결과가 문제의 내용과 일치하는가를 검증하는 단계와 (5)변경된 코드와 관련된 문서를 작성하는 단계 등의 과정을 거쳐 이루어진다.

2.1.2 유지보수의 문제점 및 비용

유지보수 단계에서 발생되는 문제점들은 개발 단계와 밀접하게 관련되어 있다. 개발 단계에서 적절한 관리와 통제가 없었거나 적합한 기법이나 도구를 사용하지 않아서 유지보수 단계에서 문제점들이 나타나게 된다. 다음은 일반적인 유지보수의 문제점들이다.

(1) 오래된 소프트웨어일수록 많은 변경이 이루어지기 때문에 변경 과정을 추적하기 어렵다.
(2) 소프트웨어와 관련된 유용한 문서가 없거나 있다하더라도 프로그램의 내용과 일치하지 않는다.

(3) 프로그래밍 형식에 대한 표준이 정립되어 있지 않고 또한, 각 프로그래머마다 프로그래밍 방식이 다르다.
(4) 소프트웨어를 개발할 때 앞으로 발생될 변경 사항을 예측하기 어렵다.
(5) 많은 프로그래머들이 유지보수 작업에 대해서 만족감을 느끼지 못한다.
(6) 소프트웨어 개발자나 유지보수자가 유동적이므로 유지보수 작업에 필요한 인력을 확보하기가 어렵다.

이러한 문제점들 때문에, 유지보수 비용이 1970년대부터 현재까지 꾸준히 증가하고 있으며 그 결과로 소프트웨어 전체 비용 중에서 개발비용보다 유지보수 비용이 훨씬 높은 비중을 차지하게 되었다. (그림 1)은 유지보수 비용의 증가 추세[5]를 연대별로 구분하여 표현한 것이다.

유지보수자가 유지보수 작업에서 가장 먼저 해야 할 일은 소프트웨어를 이해하는 것이다. 소프트웨어를 이해하는 작업은 일반적으로 3가지 활동으로 수행된다. 첫째, 관련된 문서를 읽는다. 둘째, 프로그램 즉, 원시 코드를 읽는다. 셋째, 프로그램을 실행시켜 자료를 추적하거나 논리를 추적한다. 문서를 읽는 것이 소프트웨어를 이해하는 가장 좋은 방법이지만, 유지보수자는 원시 코드를 수정해야 하기 때문에 원시 코드를 이해하는 것이 가장 중요한 작업이다. 그러나, 과거에 개발된 것은 프로그램이 표준화되어 있지 않고 요즘에 개발된 것은 규모가 매우 크고 복잡한 것이기 때문에, 유지보수자가 프로그램을 이해하는 것은 무척 힘든 일이다. 이는 유지보수 비용 중에서도 프로그램을 이해하는데 드는 비용이 가장 많이 차지하는 것과 관련된다. (그림 2)는 유지보수의 비용 중에서 프로그램 이해의 비용[6]이 차지하는 비중을 그림으로 표현한 것이다.

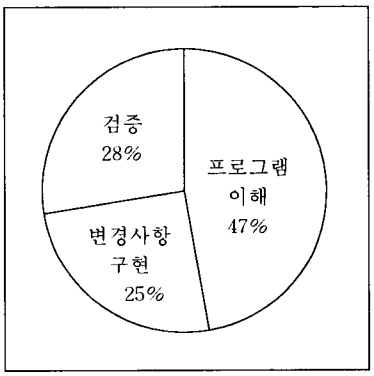


그림 2. 유지보수의 비용

Figure 2. Maintenance cost

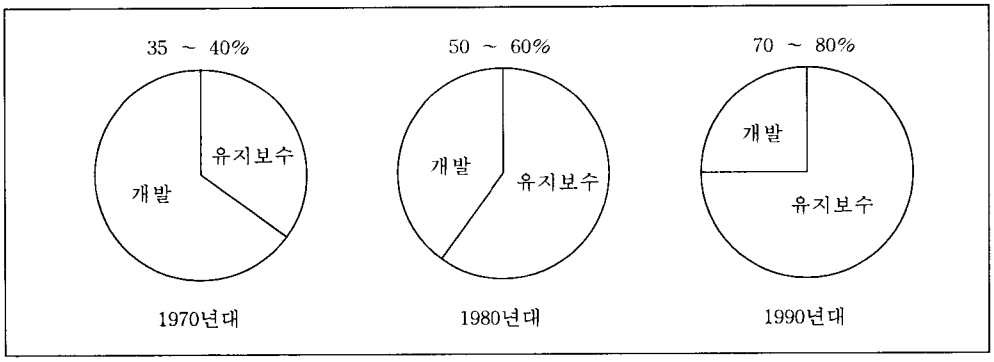


그림 1. 유지보수 비용의 증가 추세

Figure 1. The change of maintenance cost

2.2 유지보수를 위한 공학적 기법들

1990년대에 이르러 소프트웨어 생명주기 전체에 소요되는 비용 중에서 유지보수 단계에 소요되는 비용이 70-80%를 차지한다. 이는 유지보수의 어려움을 단적으로 나타내는 현상이며, 이에 따라 유지보수 비용을 절감하기 위한 많은 연구가 진행되어 왔다[7]. 유지보수 비용을 절감하기 위해서 응용 소프트웨어의 체계적인 문서화가 중요하기 때문에, 이를 지원하기 위한 자동화 도구가 필요하다. 그러나, 모든 소프트웨어의 개발에서 특정 방법론이나 도구를 이용하여 개발했다고 전제할 수 없으며, 아무리 잘 만들어진 문서가 있더라도 그것이 원시 코드로부터 유래한 것이 아니면 프로그래머들이 믿지 않으려는 성향이 있으므로, 개발된 소프트웨어를 효과적으로 유지보수하기 위해서는 원시 코드에 적용할 수 있는 도구가 필요하다.

(그림 3)은 역공학(reverse engineering), 설계회복(design recovery), 재구성(restructuring)과 재공학(reengineering)은 유지보수 단계를 지원하기 위한 공학적인 기법들이다. 이들 중에서 특히 역공학은 유지보수자에게 소프트웨어를 이해할 수 있도록 지원하며 소프트웨어를 적절하게 변경하도록 지원하는 유지보수 과정의 한 분야이다[8][11].

이 절에서는 유지보수를 지원하는 공학적인 기법들에 대한 개념과 도구들에 대해서 기술하고 특히 원시 코드로부터 설계 정보를 회복하는 작업인 역공학에 대해서 상세하게 살펴본다.

2.2.1 역공학과 설계회복

역공학은 소프트웨어가 무엇을 처리하고 어떻게 작동되는지를 표현하는 것으로서, 소프트웨어를 원시 코드보다 더 높은 추상화 수준으로 표현하기 위해 프로그램을 분석하는 과정이다. 역공학의 목적은 유지보수의 생산성을 향상시키고 소프트웨어의 이식과 변환을 지원하며 소프트웨어의 재사용 가능한 구성 요소들을 발견하도록 지원하는 것이다. 역공학 과정은 크게 3단계로 이루어진다. 첫째, 기존 소프트웨어의 원시 코드를 파싱(parsing)한다. 둘째, 소프트웨

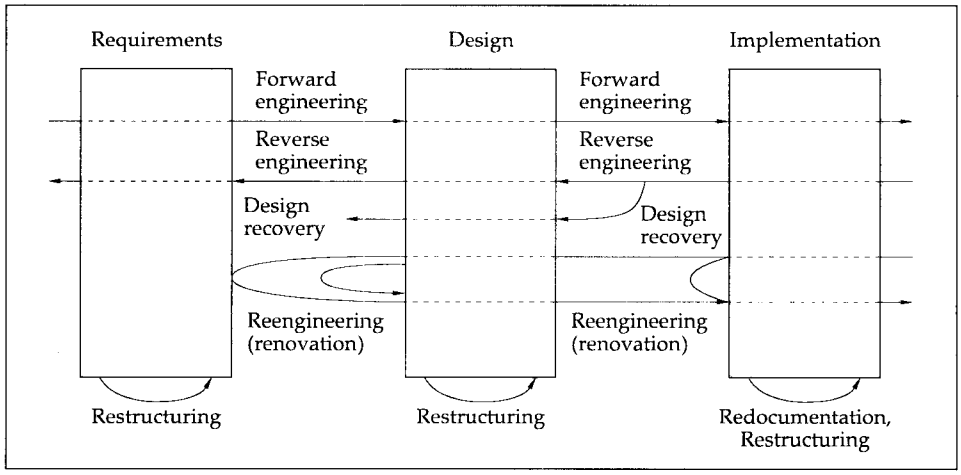


그림 3. 역공학, 설계회복, 재구성과 재공학과의 관계[8]

Figure 3. The relationship of reverse engineering, design recovery, restructuring and reengineering

어의 구성 요소들을 추출하여 저장소에 저장한다. 셋째로 저장소에 있는 회복된 정보를 그래프로 표현한다[16]. 역공학이 유지보수를 지원하는 효율적인 기법으로 많이 사용되고 있는 것은 다음과 같은 장점들이 있기 때문이다.

(1) 기존 소프트웨어에서 제공되는 문서들이나 원시 코드로부터 소프트웨어에 대한 정보를 자동적으로 인식하도록 지원한다.
(2) 기존 소프트웨어의 자료와 논리를 분석하기 위한 효과적인 방법을 제공한다.
(3) 기존 소프트웨어들에 대한 설계 정보를 저장소에 축적시킨다.

그러나, 역공학 기법만으로는 아직도 크고 복잡한 소프트웨어에 대해서 유지보수 작업의 완전한 자동화를 지원하지 못하고 있다.

설계 회복은 새로운 소프트웨어를 만들지 않고 기존 소프트웨어를 변경하지 않으면서 기존 소프트웨어의 이해를 지원하기 위해 정보를 추출해 내는 역공학의 한 분야이다.

2.2.2 재구성과 재공학

재구성은 소프트웨어의 유지보수성과 생산성을 향상시키기 위해 자료명, 자료 정의와 프로그램의 논리 구조를 표준화하는 과정이다. 단, 소프트웨어의 기능을 변화시키지 않는다. 재구성의 목적은 비구조적인 프로그램이나 정형화되지 않은 문서들을 구조적인 프로그램이나 정형화된 문서들로 변환하므로서 소프트웨어나 프로그램 자체를 이해하기 쉽도록 지원하는 것이다. 재구성은 프로그램이나 프로그램의 논리를 재구성하는 것과 자료를 재구성하는 것과 같이 두 가지 형태로 구성된다. 재공학은 기존 소프트웨어의 기능과 구조를 변경시키기 위해 역공학과 순공학을 수행하는과정이다. 재공학의 목적은 기존 소프트웨어에 대한 이해도를 높이고 공통적인 소프트웨어 구성 요소들을 발견하여, 기존 소프트웨어의 재개발이나 새로운 소프트웨어의 개발에서 이들을 재 사용할 수 있도록 하는 것이다. 그리고, 소프트웨어 개발에 필요한 시간을 줄이고 잠재된 위험을 인식하도록 하는 것도 포함한다.

2.3 사례 연구

역공학 도구들에는 2가지 유형이 있다. 하나는 자료 역공학 도구이고 다른 하나는 논리 역공학 도구이다. 자료 역공학은 기존 소프트웨어를 이해하거나 새로운 기술을 적용하는데 사용될 수 있다. 예를 들면, 기존 데이터베이스를 수정하는 것뿐만 아니라 새로운 데이터베이스관리시스템으로 이식하는 것에도 사용될 수 있다. 논리 역공학은 원시 코드로부터 처리 절차와 관련된 설계 명세를 추출하여 저장소에 적재하고 이 정보를 이용하여 모듈간의 호출 관계, 모듈의 의사 코드 등과 같은 설계 문서를 생성하는 것이다. 역공학의 사례 연구로는 첫째, FORTRAN 언어로 작성된 기존 프로그램을 ADA 언어로 변환하고 이에 따른 문서를 갱신하기 위해 역공학을 적용한 것이 있다[9]. 둘째, 원시 코드로부터 설계 명세서를 추출하기 위해 역공학 기법을 적용한 사례가 있다[12]. 셋째로는, 대규모 소프트웨어의 구조를 파악하기 위해 역공학을 적용한 사례로 RIGI 시스템이 있다[16]. 그 밖에 각 개체(entity)에 영향을 주는 논리를 추상화하기 위해 역공학을 적용한 RECAST가 있고[11] 모듈들 간의 자료 흐름을 파악하기 위해 역공학 기법을 적용한 DATA_Tool이 있다[10]. (그림 4)는 역공학과 더불어 재구성 및 재공학의 도구들이 갖는 공통적인 구조를 보여 주고 있다. 기존 소프트웨어는 이러한 도구들에 의해 새로운 관점의 산출물들로 생성된다.

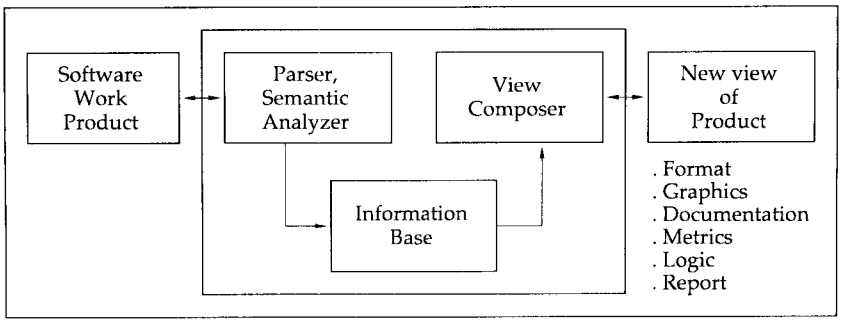


그림 4. 역공학, 재구성, 재공학 도구의 기본 구조

Figure 4. The framework of reverse engineering, restructuring and reengineering tools

2.4 구조적 설계 방법론

구조적 분석 및 설계 방법론(structured analysis and design methodology)은 데이터 구조 중심의 Warnier/Orr 방법론 및 Jackson 방법론과 데이터 흐름 중심의 Yourdon 구조적 방법론이 있다.

2.4.1 구조적 설계

구조적 분석 및 설계 방법론은 문제의 정의를 해결의 정의로 변환하기 위해 사용되는 소프트웨어 개발 기법들 중의 하나로서, 복잡하고 대규모인 시스템을 기능 중심으로 분할하여실행 가능한 작은 모듈로 만드는 방법이다[1][12]. 이 방법론은 분석과 설계로 구분된다.

구조적 분석 과정이 종료된 후, 구조적 설계 과정은 모형화 도구인 구조도, 설계 자료 사전, 프로세스 명세서를 사용한다. 이 도구들 중에서 구조도는 시스템의 단위 기능을 블랙 박스(black box)로 분할하여 모듈로 표현하고 그들 간의 인터페이스(interface)를 계층 구조로 도식화한 것으로 설계 명세서에서 가장 중요한 산출물 중의 하나이다. (그림 5)와 같이 구조도에서는 사각형(box)으로 모듈을 표현하고 화살표(arrow)를 이용하여 이 모듈간의호출 관계인 호출/피호출 관계로 표현하고 있다. 그리고, 흰점이 있는 화살표를 이용하여 자료 흐름과 흑점이 있는 화살표를 이용하여 제어 흐름을 표현한다. 구조도 상에서 모듈은 특정 처리를 수행하는 명령문의 집합으로서 다음과 같은 4가지 기본 속성을 갖는 프로그램으로 정의할 수 있다.

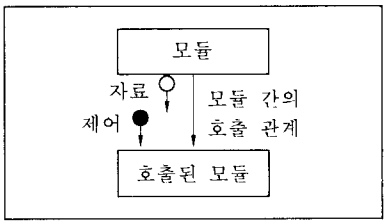


그림 5. 구조도

Figure 5. Structure chart

(1) 입력자료 : 블랙 박스의 기본 속성으로 모듈의 입력자료를 명시한 것이다. 호출자로부터 들어 온 것으로 READ, ACCEPT문이 사용된다.
출력자료 : 블랙 박스의 기본 속성으로 모듈의 출력자료를 명시한 것이다. 호출자로나가는 것으로 WRITE, DISPLAY 문이 사용된다.

(2) 처리기능 : 모듈의 처리기능을 설명하는 고유 명칭을 가진다. 수행 방식 및 내부자료를 표현하기 위한 제어가 포함된 것이며 패러그래프(paragraph)가 해당된다.

(3) 호출기능 : 호출자에서 피호출자로 제어를 넘겨 수행한 후 호출자를 계속 수행하는 모듈로 CALL문이 사용된다.

(4) 라이브러리 : 이미 구현되어 있거나 단일 명령문으로 구현 가능한 모듈로 SORT, SEARCH문이 사용된다.

모듈간의 호출관계는 화살표를 이용하여 호출/피호출 모듈 간의 관계를 표현하고 전달되는 자료와 범위에 의해 결정된다. 모듈간에 기능을 연결하기 위해 전달되는 자료는 모듈 내의 참조관계를 보는 것으로서 한 프로시저 내의 첫번째 발생한 시점으로부터 가장 마지막에 참조된 시점까지 살아 있는 자료라 가정한다.

2.4.2 구조적 프로그래밍

구조적 프로그래밍은 1969년에 E. W. Dijkstra에 의해서 소개되었다. 이것의 기본 목적은 적은 비용을 들여 양질의 프로그램을 작성하는 것이고 개념은 프로그램의 논리 구조를 단순화 하여 프로그래밍하는 것이다. 즉, 프로그램의 논리를 3가지 기본 구조인 순차, 선택, 반복 구조로 표현한다. 이것은 구조적 설계 방법론에 포함된 소프트웨어 개발 기법이며, 이 기법을 적용한 원시 코드는 구조적 프로그램이 된다. 구조적 프로그램은 하나의 입력점과 하나의 출력점을 갖는 제어 구조들의 집합으로 구성된 프로그램이며, 이 제어 구조들은 두 개 이상의 연속된 명령어들에 대한 순차 구조, 하나 이상의 연속된 명령어에 대한 조건적 선택 구조와 연속된 명령어들의 반복 구조를 포함한다[1]. (그림 6)은 관련 명령어들에 대한 구조적 프로그램의 논리 구조를 표현한 것이다.

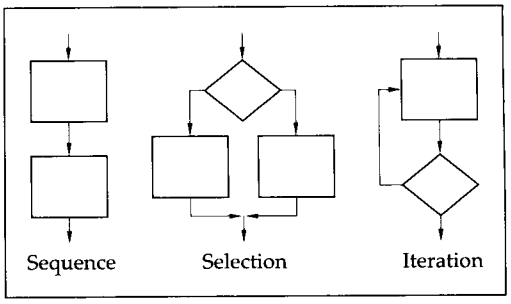


그림 6. 구조적 프로그래밍 기법의 구조

Figure 6. The framework of structured programming

(1) 순차 구조(sequence)는 프로그램의 문장들이 원시 코드에 나타난 순서대로 수행된다. 예, READ 문, WRITE문, COMPUTE문 등

(2) 선택 구조(selection or condition)는 조건을 검사한 후, 조건이 참이면 참인 블록이 수행되고, 조건이 거짓이면 거짓인 블록들이 수행된다. 예, IF-THEN-ELSE문, EVALUATE-WHEN문

(3) 반복 구조(iteration)는 블록이 먼저 수행된 다음에 조건이 검사된다. 만약, 조건이 참이면 반복은 종료되고 다음의 순차적인 명령어들이 수행된다. 만약, 조건이 거짓이면 반복 안의 블록이 수행된다. 예, PERFORM-UNTIL문, PERFORM문

Ⅲ. 구조도 생성 시스템의 설계

3.1 구조도 생성 시스템

구조도 생성 시스템은 COBOL 프로그램으로부터 전체 프로그램의 구조, 프로그램 내의

논리 구조와 프로그램의 입출력 자료를 파악할 수 있는 설계 정보를 추출하고 회복된 설계 정보로부터 전체 프로그램 구조도와 모듈 구조도를 생성하는 것이다. 또한, 각 기능과 절차를 의미있는 명칭과 명확한 구조로 수정하는 작업을 포함한다. 이 시스템의 주요 목적은 유지 보수자가 프로그램을 쉽게 이해할 수 있도록 지원하는 것이다. 이 시스템은 코드 분석의 일부 기능과 재공학의 일부 기능들을 즉, 구조도를 생성하기 위해 필요한 기능만을, 설계한 것이다.

3.1.1 구조도 생성 시스템의 구성

이 시스템은 (그림 4)의 역공학 도구의 기본 구조에 근거하여 구성한 것이다. 구조도 생성 시스템은 전체적으로 사용자 인터페이스, 코드 분석기, 구조도 생성기와 저장소로 구성되어 있다. 코드 분석기는 사용자 인터페이스를 통하여 COBOL 프로그램을 입력받고 이를 분석하고 분석된 정보를 저장소에 적재한다. 그러면, 구조도 생성기는 저장소의 정보를 입력 받아 구조도를 생성하고 또한, 사용자 인터페이스를 통하여 구조도를 편집할 수 있도록 한다. (그림 7)은 구조도 생성 시스템의 전체 구성을 표현한 것이다.

3.1.2 구조도 생성 시스템의 기능

구조도 생성 시스템은 2가지 주요 기능 즉, 코드 분석 기능과 구조도 생성 기능으로 구성되어 있다. 그리고, 구조도 생성 기능은 의미 그대로 구조도를 생성하는 기능과 편집하는 기능으로 구성된다.

(1) 코드 분석(code analysis) : 전체 프로그램의 호출 관계, 프로그램 내의 논리 구조와 프로그램의 입출력 자료에 대한

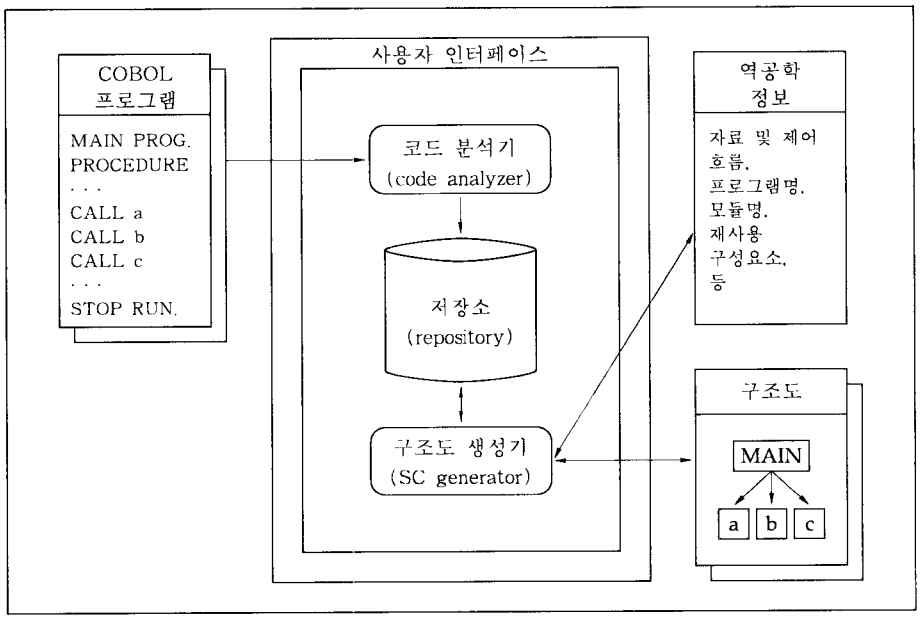


그림 7. 구조도 생성 시스템의 구성도[1][2]

Figure 7. The architecture of structure chart generation system

정보를 추출하기 위해, 기존 COBOL 프로그램을 분석하여 각각 프로그램 명세 테이블(program specification table)과 모듈 명세 테이블(module specification table)을 생성하고 저장소에 적재한다.

(2) 구조도 생성(structure chart generation) : 구조도 생성 기능은 저장소로부터 프로그램 명세 테이블과 모듈 명세 테이블을 읽어 입출력 자료의 흐름을 파악할 수 있는 프로그램 구조도(program structure chart)와 모듈 구조도(module structure chart)를 생성하는 것이다. 또한, 구조도 편집 기능은 생성된 구조도를 보기 좋게 편집하고 프로그램과 모듈의 명칭들을 의미가 있는 명칭으로 수정하여 이 수정된 정보를 저장소에 적재하는 것이다.

저장소(repository)는 각 기능들로부터 생성되거나 수정된 정보를 적재하는 장소로서 파일로 구성되어 있다.

3.2 논리 구조와 자료 흐름의 자료 모델링

이 절에서는 전체 프로그램의 구조, 프로그램 내의 논리 구조와 프로그램의 자료 구조를 식별하기 위하여 자료를 모형화하는 과정을 제시한다. 먼저, 전체 프로그램을 분석하여 프로그램간의 호출 관계를 식별하고 각 프로그램의 논리 구조를 식별한다. 또한, 프로그램간의 상호관계를 연결해 주는 입출력 자료를 식별한다. 둘째, 설계 정보에 대해 각각의 프로그램 명세 테이블과 모듈 명세 테이블을 생성하여 저장소에 적재한다. 셋째, 프로그램 명세 테이블로부터 전체 프로그램의 구조도를 생성하고 선택된 프로그램의 모듈 명세 테이블로부터 모듈 구조도를 생성한다. 또한, 각 프로그램의 입출력 흐름을 자세하게 파악할 수 있도록 입출력 자료를 표시한다. 넷째, 구조도 편집 기능을 이용하여 작성된 구조도를 수정하고 이에 따라 각 테이블을 수정하여 저장소에 보관한다.

3.2.1 프로그램 구조와 입출력 자료의 표현 방법

대부분의 응용 소프트웨어는 수많은 프로그램들로 구성된다. COBOL로 작성된 응용 소프트웨어는 일반적으로 주 프로그램, 부 프로그램들과 시스템에서 제공하는 라이브러리들로 구성된다. 본 논문에서는 다음과 같은 방법으로 단위 프로그램을 식별한다.

(1) 주 프로그램(main program)
(2) 부 프로그램(subprogram)
(3) 라이브러리(library)

프로그램들 간의 호출에 관한 내용을 구조도로 표현하게 되면, 프로그램의 호출 관계가 계층 구조로 표현된다. 계층 구조는 수직적인 구조로서 최 상위 계층에 있는 하나의 사각형(단위 프로그램)을 중심으로 이루어진다. 최 상위 계층은 수준 10^0, 그 다음의 하위 계층은 수준 10^1이 되고 바로 다음 계층은 수준 10^2이 된다. 구조도 상에서 최 상위 계층에 나타나는 사각형(수준 1)은 분석된 모든 프로그램들 중에서 다른 프로그램으로부터 호출되지 않은 프로그램이다. 즉, 주 프로그램이 된다. 한편, 구조도의 수준 1 계층에 여러 개의 사각형이 나타나게 되면 전체 프로그램들 중의 일부 프로그램만이 분석된 것이다. 즉, 프로그램 분석이 잘못된 것이다. 프로그램의 수준 번호부여 규칙은 다음과 같다.

(1) 다른 어떤 프로그램들로부터 호출되지 않는 단위 프로그램의 수준 번호는 $P_i = 1$이다. 즉, 주 프로그램이 된다.
(2) 프로그램 P_j가 주 프로그램 P_i로부터 호

출되었으면, 프로그램 P_j의 수준 번호는 P_i의 수준 번호정보 * 10^m + n이 된다. 즉, 한 피호출자의 수준 번호는 호출자의 수준 번호 * 10^m + n이 되며 다른 피호출자의 수준번호는 호출자의 수준 번호 * 10^m + n이 된다. m(m = 0,1,2...)은 호출 수준의 깊이를 나타내며, n(n = 1,2,3...)은 같은 수준에서 호출된 프로그램의 순서를 나타낸다. 단, 자기 자신이 자신을 호출하는 경우는 제외한다.

구조적 프로그램은 순차, 선택과 반복 구조인 제어 구조를 가지고 있는 것이 특징이다. 이것은 전체 프로그램의 제어 구조를 파악하는데 이해를 돕는다.

(1) IF THEN ELSE END-IF 문 내에서 호출되는 프로그램은 선택 구조로서 C(condition or selection)로 표기한다.
(2) EVALUATE WHEN END-EVALUATE 문 내에서 호출되는 프로그램은 다중 선택 구조로 M(multiple selection)으로 표기한다.
(3) PERFORM UNTIL END-PERFORM 문 내에서 호출되는 프로그램은 반복 구조로서 I(iteration)로 표기한다.
(4) (1), (2), (3) 이외에 호출되는 프로그램이나 라이브러리는 순차 구조로서 S(sequence)로 표기한다.

또한, 프로그램간을 연결해 주는 전달 자료를 파악하기 위하여 입출력 자료를 추출한다. 프로그램 A가 프로그램 A'를 호출한 경우 프로그램 A'(i, o)에서 입력 자료는 i가 되고 출력 자료는 o가 된다.

전체 프로그램에 대한 분석 작업으로 추상 정보가 추출되면 프로그램 명세에 대한 테이블이 구성되어 저장소에 적재된다. (표 1)은 프로그램 명세 테이블의 형식과 자료를 표현한 것이다.

3.2.2 논리 구조와 자료 흐름의 표현 방법

COBOL 프로그램은 영어 문장과 유사하게 작성되므로 다른 고급 언어로 작성된 프로그램보다 더 읽기 쉽다. 그러나, 아무리 읽기 쉽게 작성된 프로그램일지라도 논리가 복잡하고 규모가 클 경우에는 프로그램 내의 논리 구조를 추적하기가 쉽지 않다. 이 절에서는 프로그램내의 논리 구조와 자료 흐름을 추상 정보로 표현하는 방법에 대해서 논의하겠다. COBOL 프로그램에서 프로그램 내의 논리 구조(모듈)를 추출하는 방법은 PERFORM 문을 중심으로 이루어진다.

(1) PROCEDURE DIVISION 내의 SECTION 문
(2) PROCEDURE DIVISION 내에서 처음으로 나타나는 패러그래프(paragraph)
(3) PERFORM END-PERFORM 문과 PERFORM END-PERFORM 문 내에 포함된 PERFORM END-PERFORM 문

표 1. 프로그램 명세 테이블

Table 1. Program specification table

Program Number	Level Number	Program Name	Control Type	I · O Type		Comment
				Variable	Type	
1	1	main-prog	s			main
1	11	sub-prog	s	v_rec	i,o	sub

(4) IF THEN ELSE END-IF 문 내에 포함된 PERFORM END-PERFORM 문
(5) EVALUATE WHEN END-EVALUATE 문 내에 포함된 PERFORM END-PERFORM 문
(6) PERFORM UNTIL END-PERFORM 문 내에 포함된 PERFORM END-PERFORM 문

3.2.1절에서 프로그램의 호출 관계를 표현하는 것과 마찬가지로 프로그램 내의 논리를 계층 구조로 표현할 수 있다. 즉, 최 상위 모듈은 PROCEDURE DIVISION에서 처음에 나타나는 섹션명(section-name)이나 패러그래프명(paragraph-name)이 된다. 그리고, 섹션과 패러그래프 내에 포함된 PERFORM 문들을 구분하여 다음 계층의 모듈로 상호 관계를 표현한다. 모듈의 명칭은 섹션명이나 패러그래프명 혹은 PERFORM 문의 프로시저명(procedure-name)이 된다. 각 모듈의 수준 번호부여 규칙은 다음과 같다.

(1) 처음 나타나는 섹션이나 패러그래프의 수준 번호는 1이 된다. 수준 번호는 $M_i = 1$이다.
(2) 모듈 M_j가 주 모듈 M_i에 포함되었으면, 모듈 M_j의 수준 번호는 M_i의 수준 번호 $* 10^m + n$이된다. 즉, 한 피호출자의 수준 번호는 호출자의 수준 번호 $* 10^m + n$이 되며 다른 피호출자의 수준번호는 호출자의 수준 번호 $* 10^m + n$이 된다. $m(m = 0,1,2...)$은 호출 수준의 깊이를 나타내며, $n(n = 1,2,3...)$은 같은 수준에서 호출된 모듈의 순서를 나타낸다.

프로그램 내의 논리 구조를 파악하는 것은 전체 프로그램의 제어 구조를 파악하는 것과 유사하다. 즉, 프로그램 내의 논리 구조는 한 프로그램을 모듈 단위로 분할하여 구체적으로 표현하기 때문에 프로그램의 구조를 파악하도록 지원한다.

(1) IF THEN ELSE END-IF 문 내에 포함되는 모듈은 선택 구조로서 C(condition or selection)로 표기한다.
(2) EVALUATE WHEN END-EVALUATE 문 내에 포함되는 모듈은 다중 선택인 M(multiple selection)으로 표기한다.
(3) PERFORM UNTIL END-PERFORM 문 내에 포함되는 모듈은 반복 구조로서 I(iteration)로 표기한다.
(4) (1), (2)와 (3) 이외에 호출되는 모듈은 순차 구조로서 S(sequence)로 표기한다.

자료 흐름에 대한 분석은 입출력과 관련된 예약어가 나오면 해당 예약어의 문법에 따라 입력 자료와 출력 자료로 구분하여 추출한다. 입출력 자료의 변수를 포함하고 있는 COBOL의 예약어는 (표 2)와 같다.

표 2. 입출력 관련 예약어

Table 2. The reserved word related input/output data

예 약 어	입출력 구분
ACCEPT, READ	입력
DISPLAY, WRITE	출력
ADD, SUBTRACT, MULTIPLY, DIVIDE, COMPUTE, MOVE, SET	입출력

프로그램에 대한 분석 작업으로 추상 정보가 추출되면 모듈 명세에 대한 테이블이 구성되어 저장소에 적재된다. (표 3)은 모듈 명세 테이블을 구성하는 형식과 자료를 표현한 것이다.

3.2.3 구조도의 표현 방법

설계 회복 정보의 표현을 위한 기초 단계로 목표 대상인 프로그램(혹은, 모듈) 기능과 프로그램들(혹은, 모듈들) 간의 호출 관계를 정의하여야 한다. 단위 프로그램은 사각형으로 표현하고 내부에 프로그램 명을 기술한다. 그리고, 프로그램들 간의 호출 관계는 화살표로 표현하며, 입출력 자료의 흐름은 흰점이 있는 화살표를 이용하여 표현한다.

(1) 구조도의 표현에서 각 프로그램(혹은, 모듈)은 사각형으로 그리고 기능은 사각형 내에 프로그램명(혹은, 모듈명)으로, 호출 관계는 긴화살표로 표현하고 짧은 화살표를 이용하여 자료의 흐름을 표현한다.

(2) 프로그램이나 모듈 명세 테이블에서 수준 번호가 1인 프로그램이나 모듈은 구조도 상에서 최 상위 계층에 표현한다.

(3) 프로그램의 제어 흐름과 모듈의 논리 구조는 프로그램 명세 테이블이나 모듈 명세 테이블의 제어 형식에 따라 표현한다.

(4) 프로그램의 자료 흐름은 명세테이블의 입출력 형식에 따라 표현한다.

위와 같은 규칙에 따라, 모든 구조도는 반드시 상위 계층을 중심으로 하위 계층을 표현한다. 선택 구조는 단순과 다중 선택으로서 검은 사각형으로 조건을 표시한다. 반복 구조는 원이 있는 화살표를 이용하여 반복을 표현한다. 표시가 없는 것은 순차 구조를 의미한다프로그램이나 모듈의 기능, 구조와 자료에 대한 표현은 다음의 구조도로 나타낸다.

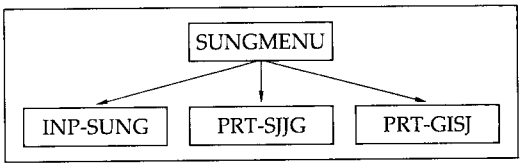


그림 8. 순차 구조

Figure 8. Sequence

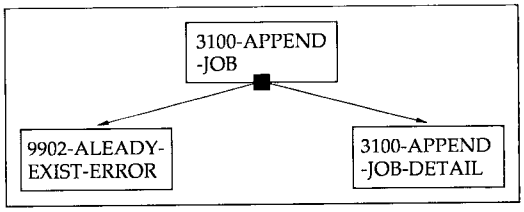


그림 9. 단순 선택 구조

Figure 9. IF THEN ELSE END-IF.

표 3. 모듈 명세 테이블

Table 3. Module specification table

Program Number	Program Name	Module Number	Module Name	Control Type	I · O Type		Comment
					Variable	Type	
1	MAINPROG	1	000-step-process	i			main
1	MAINPROG	11	111-input-process	i	var1	i	main
1	MAINPROG	12	200-main-process	s	var2	i,o	sub
1	MAINPROG	13	300-output-process	i	var1	o	main

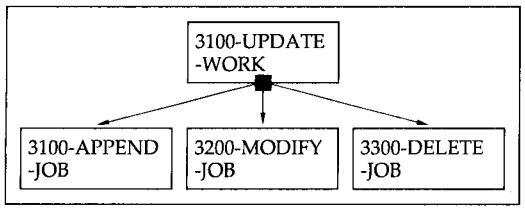


그림 10. 다중 선택 구조

Figure 10. EVALUATE WHEN END-EVALUATE.

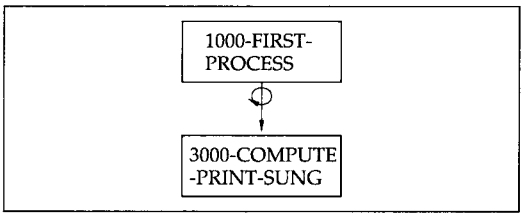


그림 11. 반복 구조

Figure 11. Iteration

Ⅳ. 구현 및 실험

본 장에서는 구조도 생성 시스템의 구현 환경에 대해서 기술하고, 주요 화면과 코드 분석과 구조도 생성 및 편집 기능들에 대해서 설명한다. 구조도 생성 시스템은 486PC의 DOS V6.0 환경 하에서 Borland C++로 개발하였다. 그리고 실험예제로서 RM COBOL로 작성된 프로그램[3]을 사용하였다.

구조도 생성 시스템은 코드 분석기, 구조도 생성기와 구조도 편집기로 구성되어 있다. 이 시스템은 사용자 인터페이스를 위하여 (그림 12)와 같은 초기 화면을 제공한다. 그리고, 이 시스템은 코드 분석과 구조도 생성 등의 주요 메뉴와 세부 메뉴로 구성되어 있다.

4.1 코드 분석기

코드 분석기는 COBOL로 작성된 프로그램들을 입력받아 프로그램 명세 파일과 모듈 명세 파일을 생성하는 기능이다. 코드 분석 메뉴에서 불러오기 메뉴를 선택하면, RM COBOL로 작성된 프로그램의 목록이 나타난다. 프로그램을 분석하기 위해서는 이 목록들 중에서 원하는 프로그램을 선택해야 한다. 만약, 선택된 프로그램이 분석된 코드라면, 이 코드 분석

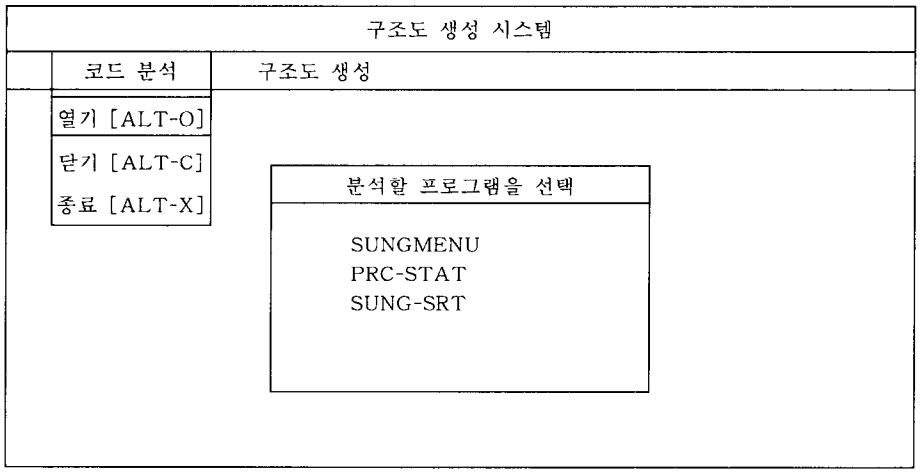


그림 12. 구조도 생성 시스템의 초기 화면

Figure 12. The initial screen of structure chart generation system

기는 이미 분석된 프로그램이라는 메시지와 함께 프로그램 명세 파일의 명칭을 화면에 나타낸다.

분석된 것이 아니라면, 선택된 프로그램은 코드 분석 과정을 거치게 되고 그 결과로 프로그램 명세 파일과 모듈 명세 파일이 생성된다. 이 파일들은 바로 저장소에 적재된다. (표 4)와 (표 5)는 RM COBOL로 작성된 프로그램들을 코드 분석기에서 작업하여 나타난 결과이다.

4.2 구조도 생성기

구조도 생성기는 저장소로부터 프로그램 명세 파일과 모듈 명세 파일을 입력받아 전체 프로그램 구조도와 모듈 구조도를 생성하는 기능이다. 구조도 생성 메뉴는 의미 그대로 구조도를 생성하는 기능과 구조도를 수정하는 편집 기능으로 구성되어 있다.

먼저, 구조도를 생성하기 위해서는 분석된 프로그램 명세 파일을 선택하여야 한다. (그림 13)은 구조도 생성기의 기본 화면이다. 이 기본 화면에서 파일 불러오기를 선택하면, 프로그램 명세 파일들의 목록이 나타난다. 선택된 프로그램 명세 파일에 대해서 구조도 생성기는 이 파일을 입력으로 전체 프로그램에 대한 구조도를 생성한다 (그림 14). 그리고, 전체 구조도 상에서 원하는 프로그램에 대한 모듈의 구조도를 생성하기 위해서, 원하는 상자(프로그램)를 선택한 다음 구조도 편집 메뉴의 자세히 보기를 선택한다(그림 14).

표 4. 프로그램 명세 파일의 구성

Table 4. Program specification file

Program Number	Level Number	Program Name	Control Type	I · O Type		Comment
				Variable	Type	
1	1	SUNG-MENU	s			main
1	13	INP-SUNG	s			i
1	11	PRT-SJJG	s	jumsu_rec	i	
				compute_rec	i	
1	13	PRT-GISJ	s	o_jumsu_rec	o	

표 5. 모듈 명세 파일의 구성

Table 5. Module specification file

Program Number	Program Name	Module Number	Module Name	Control Type	I · O Type		Comment
					Variable	Type	
1	PRT-SJJC	1	0000-print-sungjukpyo	s			main
1	PRT-SJJC	11	1000-first-process	s	sungjuk_rec	i	
1	PRT-SJJC	111	3000-compute-sungjuk	i			
1	PRT-SJJC	12	4000-last-process	s	sungjuk_rec	o	

구조도 생성 과정에 따라, 프로그램 명세 파일로부터 생성된 구조도의 예가 (그림 14)에 제시되어 있다.

4.3 구조도 편집기

구조도 생성기에 포함된 구조도 편집기는 생성된 각 구조도를 수정하고 수정된 내용에 따라 프로그램 명세 파일과 모듈 명세 파일을 생성하여 저장소에 적재하는 기능이다. 이 편집기는 구조도 입력, 수정, 이동과 삭제 메뉴로 구성되어 있다(그림 14). 이 기능은 유지보수자의 판단에 의해 생성된 구조도를 명확한 구조를 갖는 새로운 구조도로 수정하는 것이

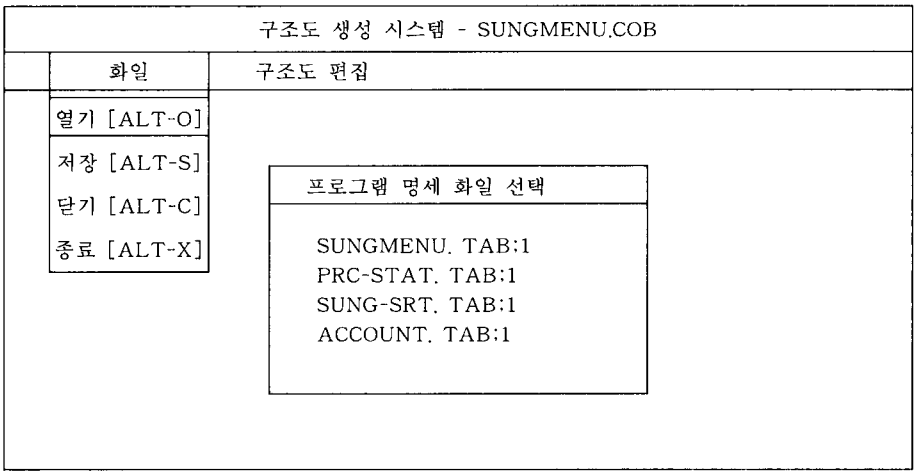


그림 13. 구조도 생성기의 화면

Figure 13. The structure chart generator

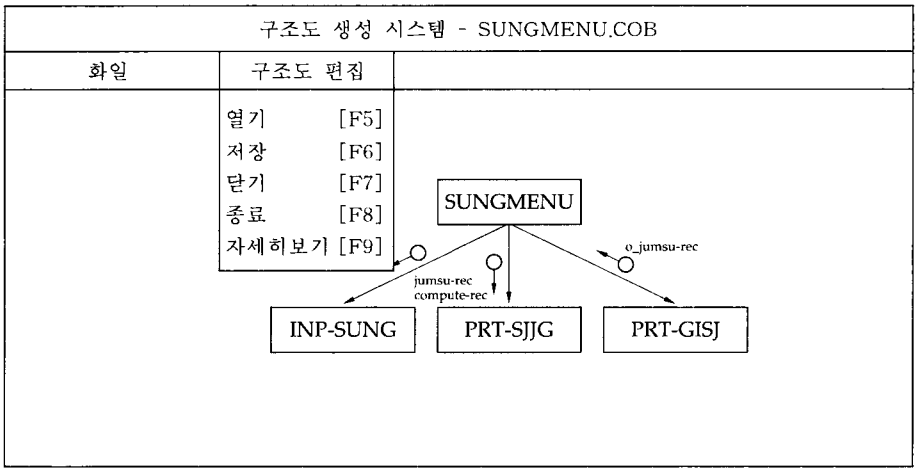


그림 14. 구조도 편집기의 화면

Figure 14. The structure chart editor

다. 또한, 의미를 정확하게 하기 위해서 기능의 명칭을 수정할 수 있다. (그림 15)는 전체 프로그램 구조도와 선택된 프로그램에 대한 모듈 구조도를 함께 자료의 흐름을 보여주고 있다. 또한, 수정한 내용을 저장소에 적재하기 위해 파일 메뉴에서 저장하기가 있다. 이 기능은 생성된 각 파일들을 버전별로 관리하므로 프로그램의 재분석 과정과 재실행 과정을 최소화 하였다.

4.4 실험 결과

구조도 생성 시스템의 정확성을 검증하기 위하여, "구조적 COBOL과 응용"의 프로그램 예제들 중에서 20개를 선택하여 시스템에 적용시켜 보았다. 그 결과, 전체 프로그램 구조도와 모듈 구조도를 정확하게 생성하였다. 실험된 결과가 문제가 없다고 판단되었을 때, 실제로 운용 중인 5개의 프로그램들을 적용하여 구조도를 생성하였다. 그 결과 역시 정확한 구조도를 생성하였다. 한 예로 구조도 생성 시스템을 적용한 실제 프로그램과 결과가 부록에 첨부되어 있다.

구조도 생성 시스템을 이용하여, 원시 코드로부터 정확하게 생성된 구조도는 유지보수자들에게 프로그램을 전체적으로 혹은 구체적으로 이해하는데 상당한 도움이 될 것이라 기대된다. 또한, 유지보수자가 다른 시점에서 프로그램을 재분석하거나 다른 유지보수자들이 그 프로그램을 분석할 때, 구조도 편집 기능을 이용하여 의미있는 명칭과 보다 명확한 구조로 수정된 구조도는 프로그램을 보다 더욱 쉽게 이해하도록 지원할 것이다. 그러나, 전체 프로그램의 구조, 프로그램 내의 논리 구조와 자료 흐름만으로 모든 소프트웨어를 완벽하게 이해할 수 없다. 그리고, 프로그램과 모듈에 대한 구조가 정형화된 조건에서 출발하였기 때문에, 이 시스템 역시 매우 복잡하고 대규모인 시스템에 대한 구조도를 생성할 수 없다는 문제점이 여전히 남아 있다.

V. 결 론

본 논문에서는 유지보수를 용이하게 하기

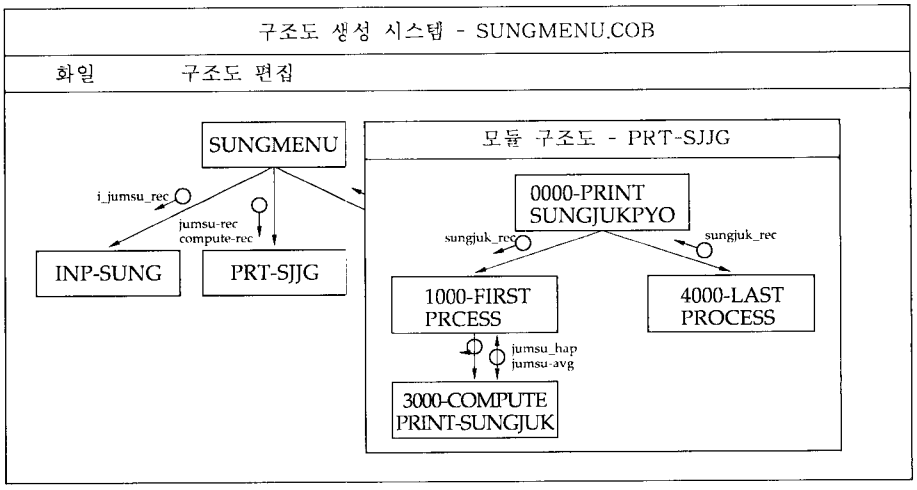


그림 15. 프로그램 및 모듈 구조도의 실예

Figure 15. The example of structure chart

위해 특히, 프로그램을 이해하기 쉽도록 지원하기 위해, 역공학 접근 방법을 이용하여 COBOL 언어로 작성된 프로그램들로부터 설계 정보를 회복하는 방법과 회복된 설계 정보로부터 전체 프로그램의 구조와 프로그램 내의 논리 구조와 프로그램의 자료 흐름을 표현하는 방법에 대해서 기술하였다. 또한, 본 논문에서 제시된 방법들을 실현하기 위해 코드 분석 기능과 구조도 생성 기능을 갖는 구조도 생성 시스템을 구현하였다.

이 시스템의 시험을 위해, RM COBOL로 작성된 예제들을 대상으로 실험하였지만 기타 COBOL 프로그램들에도 적용될 수 있다는 사실을 알았다. 이와 같이, 원시 코드로부터 정확하게 생성된 구조도는 프로그램을 전체적 혹은 구체적으로 이해하는데 중요한 정보가 될 것이다. 또한, 유지보수자가 다른 시점에서 프로그램을 재분석하거나 혹은, 다른 유지보수자 들이 그 프로그램을 분석할 때, 구조도 편집 기능을 이용하여 명확한 구조로 수정된 구조도는 유지보수자가 프로그램을 보다 더욱 쉽게 이해하도록 지원할 것이다. 구조도 만으로 프로그램 이해를 완벽하게 지원하지 못하기 때문에, 설계 단계를 완전히 지원하기 위한 작업으로서 자료 흐름과 자료 구조를 회복하는 방법과 원시 코드로부터 의사 코드를 생성하는 방법에 대해서 연구되어야 한다. 그리고, 원시 코드로부터 설계 문서를 생성하는 방법과 이 정보를 체계적으로 저장소에 관리하는 방법에 대해 지속적으로 연구되어야 한다.

부 록

1. 프로그램(성적처리 프로그램)

```
IDENTIFICATION      DIVISION.
PROGRAM-ID.         SUNGJUK.
ENVIRONMENT         DIVISION.
INPUT-OUTPUT        SECTION.
FILE-CONTROL.
    SELECT SUNGJUK-RESULT-FILE ASSIGN TO OUTPUT "SUNGJUK.REP"
           ORGANIZATION IS LINE SEQUENTIAL.
DATA                DIVISION.
FILE                SECTION.
FD  SUNGJUK-RESULT-FILE LABEL RECORD IS STANDARD.
01  SUNGJUK-RESULT-REC  PIC X(79).
WORKING-STORAGE SECTION.
01  PROMPT-TITLE.
    03  FILLER              PIC X(19) VALUE " 학 번 및 점 수 입 력 ".
    03  PROMPT-LINE         PIC X(19) VALUE ALL "_".
01  PROMPT-JEOMSU-REC.
    03  FILLER              PIC X(07) VALUE ALL "9".
    03  FILLER              PIC X(03) VALUE ALL SPACES.
```

```
    03  FILLER              PIC X(03) VALUE ALL "999".
    03  FILLER              PIC X(03) VALUE ALL SPACES.
    03  FILLER              PIC X(03) VALUE ALL "999".
01  GET-JEOMSU-REC.
    03  G-HAKBEON           PIC 9(07).
        88 END-COND         VALUE ZERO.
    03  FILLER              PIC X(03).
    03  G-COBOL             PIC 9(03).
    03  FILLER              PIC X(03).
    03  G-C                 PIC 9(03).
01  JEOMSU-SUM               PIC 9(05).
01  JEOMSU-AVE               PIC 999V99.
    88  A-COND               VALUE 90.0 THRU 100.0.
    88  B-COND               VALUE 80.0 THRU 89.99.
    88  C-COND               VALUE 70.0 THRU 79.99.
    88  D-COND               VALUE 60.0 THRU 69.99.
    88  F-COND               VALUE 0.0 THRU 59.99.
01  PRINT-TITLE.
    03  FILLER               PIC X(18) VALUE SPACES.
    03  FILLER               PIC X(10) VALUE "성 적 표".
01  PRINT-HEADING.
    03  PROMPT-HEADING.
        05  FILLER           PIC X(10) VALUE " 학 번 ".
        05  FILLER           PIC X(10) VALUE " COBOL ".
        05  FILLER           PIC X(10) VALUE " C ".
    03  REST-HEADING.
        05  FILLER           PIC X(08) VALUE " 합 계 ".
        05  FILLER           PIC X(08) VALUE " 평 균 ".
        05  FILLER           PIC X(08) VALUE " 등 급 ".
01  PRINT-LINE PIC X(44) VALUE ALL "-".
01  PRINT-REC.
    03  P-HAKBEON           PIC 9(7).
    03  P-COBOL             PIC ZZ9.
    03  FILLER              PIC X(3) VALUE SPACES.
    03  P-C                 PIC ZZ9.
    03  FILLER              PIC X(3) VALUE SPACES.
    03  P-JEOMSU-SUM        PIC Z(4)9.
    03  FILLER              PIC X(3) VALUE SPACES.
    03  P-JEOMSU-AVE        PIC ZZ9.99.
    03  FILLER              PIC X(3) VALUE SPACES.
    03  P-GRADE             PIC X(1).
```

```
PROCEDURE           DIVISION.
LEVEL-0             SECTION.
0000-PROCESS-SUNGJUK.
     PERFORM 1000-INIT-PROCESS.
     PERFORM 2000-GET-JEOMSU-REC.
     PERFORM UNTIL END-COND
             PERFORM 3000-COMPUTE-PRINT-SUNGJUK
             PERFORM 2000-GET-JEOMSU-REC
     END-PERFORM.
     PERFORM 4000-LAST-PROCESS.
     STOP RUN.
LEVEL-1             SECTION.
1000-INIT-PROCESS.
     PERFORM 1100-OPEN-FILE.
     PERFORM 1200-PRINT-HEADING.
     PERFORM 1300-DISPLAY-PROMPT.
2000-GET-JEOMSU-REC.
     INITIALIZE GET-JEOMSU-REC.
     ACCEPT GET-JEOMSU-REC.
3000-COMPUTE-PRINT-SUNGJUK.
     PERFORM 3100-COMPUTE-SUM.
     PERFORM 3200-COMPUTE-AVERAGE.
     PERFORM 3300-EVALUATE-GRADE.
     PERFORM 3400-MOVE-PRINT-SUNGJUK.
4000-LAST-PROCESS.
     PERFORM 4100-LAST-LINE.
     PERFORM 4200-CLOSE-FILE.
LEVEL-2             SECTION.
1100-OPEN-FILE.
     OPEN OUTPUT SUNGJUK-RESULT-FILE.
1200-PRINT-HEADING.
     WRITE SUNGJUK-RESULT-REC FROM PRINT-TITLE.
     WRITE SUNGJUK-RESULT-REC FROM PRINT-LINE.
     WRITE SUNGJUK-RESULT-REC FROM PRINT-HEADING.
     WRITE SUNGJUK-RESULT-REC FROM PRINT-LINE.
1300-DISPLAY-PROMPT.
     DISPLAY PROMPT-TITLE.
     DISPLAY SPACES.
     DISPLAY PROMPT-LINE.
     DISPLAY PROMPT-HEADING.
     DISPLAY PROMPT-JEOMSU-REC.
     DISPLAY PROMPT-LINE.
```

```
3100-COMPUTE-SUM.
     COMPUTE JEOMSU-SUM = G-COBOL + G-C.
3200-COMPUTE-AVERAGE.
     COMPUTE JEOMSU-AVE = JEOMSU-SUM / 2.
3300-EVALUATE-GRADE.
     EVALUATE TRUE
               WHEN A-COND
                    MOVE "A" TO P-GRADE
               WHEN B-COND
                    MOVE "B" TO P-GRADE
               WHEN C-COND
                    MOVE "C" TO P-GRADE
               WHEN D-COND
                    MOVE "D" TO P-GRADE
               WHEN F-COND
                    MOVE "F" TO P-GRADE
               WHEN OTHER
                    MOVE SPACE TO P-GRADE
     END-EVALUATE.
3400-MOVE-PRINT-SUNGJUK.
     MOVE G-HAKBEON TO P-HAKBEON.
     MOVE G-COBOL TO P-COBOL.
     MOVE G-C TO P-C.
     MOVE JEOMSU-SUM TO P-JEOMSU-SUM.
     MOVE JEOMSU-AVE TO P-JEOMSU-AVE.
     MOVE SUNGJUK-RESULT-REC FROM PRINT-REC.
4100-LAST-LINE.
     WRITE SUNGJUK-RESULT-REC FROM PRINT-LINE.
4200-CLOSE-FILE.
     CLOSE SUNGJUK-RESULT-FILE.
```

2. 프로그램/모듈 명세 테이블

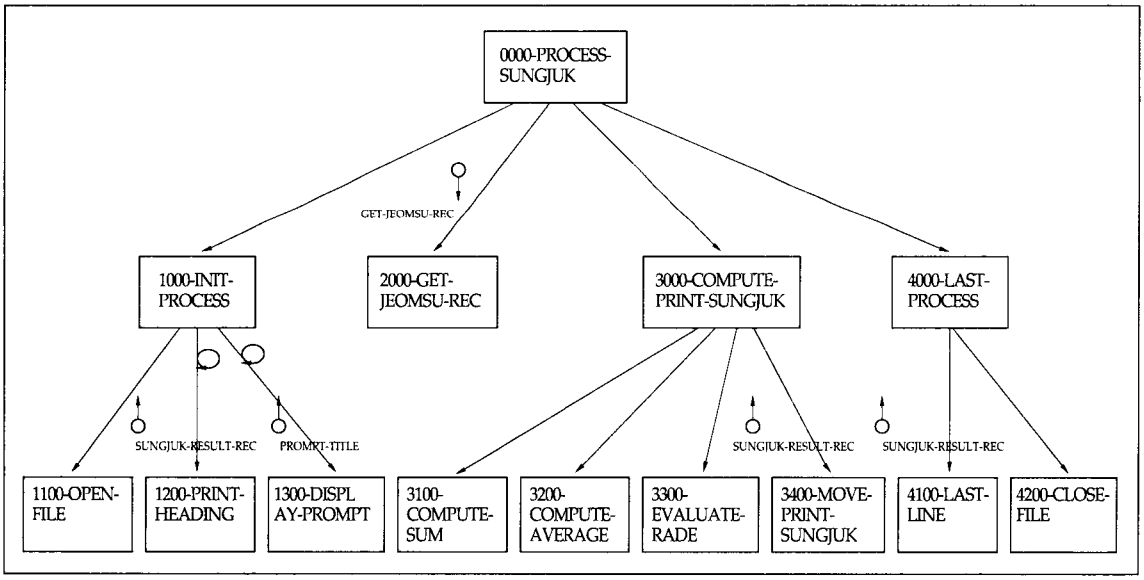
2.1 프로그램 명세 테이블

Program Number	Level Number	Program Name	Control Type	I · O Type		Comment
				Variable	Type	
1	1	SUNGJUK	S			main

2.2 모듈 명세 테이블

Program Number	Program Name	Module Number	Module Name	Control Type	I · O Type		Comment
					Variable	Type	
1	SUNGJUK	1	0000-PROCESS-SUNGJUK	S			
1	SUNGJUK	11	1000-INIT-PROCESS	S			
1	SUNGJUK	111	1100-OPEN-FILE	S			
1	SUNGJUK	112	1200-PRINT-HEADING	I	SUNG-RESULT-REC	O	
1	SUNGJUK	113	1300-DISPLAY-PROMPT	I	PROMPT-TITLE	O	
1	SUNGJUK	12	2000-GET-JEOMSU-REC	S	GET-JEOMSU-REC	I	
1	SUNGJUK	13	3000-COMPUTE-PRINT-SUNGJUK	S			
1	SUNGJUK	131	3100-COMPUTE-SUM	S			
1	SUNGJUK	132	3200-COMPUTE-AVERAGE	S			
1	SUNGJUK	133	3300-EVALUATE-GRADE	S			
1	SUNGJUK	134	3400-MOVE-PRINT-SUNGJUK	S	SUNGJUK-RESULT-REC	O	
1	SUNGJUK	14	4000-LAST-PROCESS	S			
1	SUNGJUK	141	4100-LAST-LINE	S	SUNGJUK-RESULT-REC	O	
1	SUNGJUK	142	4200-CLOSE-FILE	S			

3. 구조도



참 고 문 헌

[1] 서민호, 구조도 생성을 위한 역공학 접근방법, 숭실대학교 석사학위논문, 1994

[2] 최석환, 역공학 기법을 적용한 입출력 자료 추출 도구, 숭실대학교 석사학위논문, 1994

[3] 류성열, 김진수, 우경환, 구조적 COBOL과 응용, 생능, 1993.

[4] 우치수, 구조적 기법 소프트웨어 공학, 상조사, PP.51-65, PP.175-185, 1992.

[5] 천유식, 컴퓨터와 컴퓨터관련 시스템개발방법론, 컴퓨터엔지니어링, PP.123-136, 1991.

[6] Carma McClure, The Three Rs(Re-engineering Repository Reusability) of Software Automation, Prentice-Hall, 1992.

[7] David Frost, Software Maintenance and Modifiability, Proceedings of the 1985Phoenix Conference on Computers and Communications, 1985.

[8] Elliot J. Chikofsky and James H. Cross II, Reverse Engineering and Design Recovery : A Taxonomy, IEEE Scftware, vol.7, no.1, pp.13-17, Jan. 1990.

[9] Eric J. Byrne, Software Reverse Engineering : A Case Study, Software Practice andExperience, pp.1349-1364, Dec. 1991.

[10] G. Canfora, A. Cimitile, and U. De Carlini, A Logic Based Approach to Reverse Engineering Tools Production, IEEE Conference on Software Maintenance, Oct. 1991.

[11] Helen M. Edwards and Malcolm Munro, Abstracting the Logical Processing Life Cycle for Entities Using the RECAST Method, IEEE Conference on Software Maintenance, Sep. 1993.

[12] Hongji Yang, The Supporting Environment for A Reverse Engineering System - The Maintainer's Assistant, IEEE Conference on Software Maintenance, Oct. 1991.

[13] Meilir Page-Jones, The Practical Guide to Structured Systems Design, 2ndEdition, Prentice-Hall, 1988.

[14] Marijuana Tomic, A Possible Approach to Object-Oriented Reengineering of COBOL Programs, ACM SIGSOFT, Software Engineering Notes vol.19, no.2, Apr. 1994.

[15] Roger S. Pressman, Software Engineering : A Practitioner's Approach, 3rd Edition, McGraw-Hill, pp.663-680, 1992.

[16] Scott R. Tilley, Hausi A. Muller, and Michael J. Whitney, Kenny Wong, Domain-Retargetable Reverse Engineering, IEEE Conference on Software Maintenance, Sep. 1993.

□ 著者紹介

서 민 호

1964년 7월 19일
1989. 2. 숭실대학교 전자계산학과 공학사
1995. 2. 숭실대학교 대학원 전자계산학과 공학석사
1989. 11. ~ 1992. 11. 한국건설기술연구원
1993. 3. ~ 1994. 2. 숭실대학교 중앙전자계산소
1995. 2. ~ 현재 광주여자전문대학 사무자동화과 전임강사

※ 관심 분야 : 소프트웨어공학, 멀티미디어