Journal of Korea Multimedia Society Vol. 18, No. 12, December 2015(pp. 1528-1537)
http://dx.doi.org/10.9717/kmms.2015.18.12.1528

개요도구 : 초보 프로그래머를 위한 간편 설계도구

박 세 명†

Outlining Tool as an Easy-to-Use Designing Tool for a Novice Programmer

Se Myung Park†

ABSTRACT

In this research, we show that constructing a specific job-processing-sequences like a OPL paragraph in OPM (we call it int a new terminology 'program', which just means plan) is essential step in programming, just like a blueprint for a wood furniture or a house. As a result of our research on how to produce a specific job-processing-sequences without OPM, we found that outlining tool is the most easy-to-use tool for the required job. We show that the MS-word outlining function can be used easily as a good programming supporting tool with Visual Studio C/C++.

Key words: OPM, OPL Paragraph, Outlining Tool, MS-word,

1. 서 론

목가구작업조차도 스케치업이라는 s/w를 이용하여 3차원으로 가구를 그려본 후(모의실행과 검증과정), 정확한 치수의 재료를 준비하고 조립작업을 시작하는데 이에 못지않게 복잡한 과정을 수반하는 프로그래밍 작업에서 어떤 준비과정을 거쳐 소스프로그램을 작성하였는지 생각해보자. 모든 준비과정을 머릿속에서, 때로는 약간의 종이와 연필을 사용하여 끌쩍여보는 등의 구체적이지 못한 준비과정을 수행한 후 코딩 작업을 시작하지 않았는가? 실상 프로그래밍 작업 과정에서 대상 작업에 대한 충분한 분석작업을 위해 많은 시간이 할당되어야 함에도 불구하고 준비없는 프로그래밍 작업은 단지 비효율적인 소스프로그램의 생산 뿐 아니라, 초보 프로그래머에게는 극복하기 힘든 큰 장애로 여겨지는게 현실이다. 효과적인 프로그래밍 교육을 위해 다양한 도구를 사용하여 문제에 대한 깊은 이해가 프로그래밍에 도움이 된다는 접근법[1,2,3]과 나이수준에 적합하게 4조각, 16조각 맞추기 퍼즐에서 시작하여 단계별로 더 복잡한 퍼즐을 완성해가면서 스스로 경험에 의해서 더 복잡한 퍼즐을 맞출 수 있는 방법을 습득하기를 바라는 것과 유사한 접근법[4,5,6,7]이 있지만 이러한 접근법들은 작은 응용프로그램을 작성하는 연습을 하면서 프로그래밍 언어의 문법 활용연습을 하다보면 프로그래밍에 대한 경험과 이해가 증진되어 프로그래밍 능력을 습득하게 된다는 일반적인 프로그래밍 습득과정에 근본적인 변화를 제시하지 못하고 있다. 프로그래밍 과정에서 문제분석과정을 거친 후 코딩 작업으로 전환하기 위해서는 코딩이 가능한 상세한 수준의 문제에 대한 이해가 선행[8]되어야 하는데 단순히 작업을 효과적으로 분석해보거나, 소규묘 작업을 수행해본 경험만으로는 코딩수준의 상세한 이해를 보장하기 어렵기 때문이다.

※ Corresponding Author : Se Myung Park, Address: (621-749) 197 Inje-ro Kimhae-city Kyungsangnam-do, TEL : +82-55-320-3276, FAX : +82-55-322-3107, E-mail : cssmpark@inje.ac.kr
Receipt date : Aug. 28, 2015, Revision date : Nov. 4, 2015
Approval date : Nov. 17, 2015
† School of Computer Engineering, Inje *University
※ This research was supported by Inje Research & scholarship foundation

본 연구에서는 '프로그래밍 작업'의 의미를 "문제 분석을 통한 이해를 기반으로 코딩수준의 작업처리 절차인 프로그램(사전적의미의 작업절차)을 작성하고, 이를 선택된 컴퓨터언어로 코딩하는 작업"으로 해석하여 '프로그램작성' 작업과 '코딩' 작업은 완전히 분리된, 절차상 서로 다른 단계의 작업을 의미하도록 정의하였다. 그리고 기존의 "프로그램을 작성"한다는 말에서 소스코드를 의미하는 프로그램이라는 용어는 혼돈을 피하기 위해 소스프로그램, 소스코드, 또는 특정언어를 첨부하여 C프로그램, Java프로그램이라는 용어로 표기한다.

본 연구에서는 프로그래밍 작업에 대한 재해석을 바탕으로 프로그래밍 작업에서 목가구 작업에서의 설계도면 그리기 과정과 동일한 의미인 검증작업(모의실행)에 해당하는 '프로그램작성' 작업이 어떤 체계적인 과정을 거쳐 수행되야할 지를 연구하여 '프로그램 작성' 작업에 적합한 프로그래밍 보조도구를 제안하고자 하였다. 이를 위해 우선 기존의 프로그래밍 보조도구인 flowchart, UML [9,10,11], OPM[12,13,14]이 프로그래밍을 위한 검증과정 또는 프로그램작성을 어떻게 지원하는지 평가, 고찰할 수 있도록 평가척도(정성평가)를 제안하였는데 이러한 평가를 통해 체계적인 검증작업(모의실행, 또는 프로그램작성작업)을 지원하는 도구가 갖추어야할 의미있는 특징을 도출할 수 있었고 이러한 특징을 포함하는 보조도구(모의실험, 프로그램작성)로 개요도구(outlining tool)을 제안하였다. 개요도구는 Visual Studio개발환경과 기본적인 통합환경만을 제공하며 단계별 또는 수준별 설계과정에 대한 설명이 가능하고, 사용법을 익히기 쉽고, top-down 접근법에 따라 프로그래밍 시작단계인 개략적인 설계단계에서부터 코딩작업 직전의 구체적인 설계까지의 모든 과정을 효과적으로 검증하는 등 프로그래밍 보조도구의 핵심기능을 완벽하게 지원하고 있다.

2장은 기존 프로그램 개발도구 평가를 위해 우선 평가척도를 제시하고, 제안된 평가척도에 따라 UML [9], OPM[13], 흐름도(flowchart)를 평가하였다. 3장에서 평가척도가 제시한 개발도구가 갖추어야할 핵심요소와 기존 도구의 개선할 점을 고려하여 프로그램 개발 보조도구로 개요도구를 제안한 후, 적용사례와 고찰을 기술하고, 마지막으로 4장에서 결론을 기술하였다.

2. 기존 프로그램 개발 보조도구 평가

지금까지 연구된 다양한 프로그래밍 개발 보조도구나 코드 자동생성 도구들의 기능을 나열, 설명하는 것 만으로는 서로 다른 목적으로 개발된 이들 도구를 비교, 평가하기 어렵다. 따라서 본 연구에서는 각 보조도구들이 목가구 작업에서의 목가구 그리기 과정과 동일한 구체적인 검증과정을 바탕으로 프로그래밍 작업 수행을 지원하는지 평가하고자 한다.

2.1 평가척도 제시

프로그래밍 개발 보조도구의 평가에서는 프로그램의 준비시간과 검증작업에 소요되는 시간이 실제 프로그래밍 시간에 끼치는 영향을 평가하려는 것이 주 목적이므로, 비교척도의 구성 요소로 해당 도구를 사용할 경우에 프로그래밍작업 과정에서 프로그래머가 수행할 작업을 준비작업(도구이해, 도구 사용법 숙지 등)과 검증(설계 및 모의실행, 프로그램작성)작업, 그리고 실제 코딩 작업에서 소요되는 작업량으로 선정하였다. 각 도구의 각 요소들에 대한 평가(해당 단계에서의 작업량)는 해당 도구의 특징을 잘 나타내고 있다고 판단된다. 각 단계들의 작업량을 표시하기위해서는 각 단계의 작업량을 5단계 기준을 사용하여 나타내도록 하였다. 제시된 평가척도(EvaluationCriteria) Ec(x/y, z)는 아래와 같이 표시된다.

평가척도 : Ec(x/y, z)
- x ; 프로그래머(개발자) 준비작업, 도구이해과정, (생략가능)
- y : 프로그램검증(설계 및 모의실행)과정, 즉 프로그램 작성과정
- z : 코딩 과정

제시된 평가 척도에 따른 평가의 객관성 유지를 위해 다수의 컴퓨터전공자(10명, 석사과정5, 박사과정3, 교수2,)들이 평가에 참여하였다. 평가 결과는 Ec(1/3(2), 3)으로 표시될 수도 있는데 각 요소별 평가값은 평가에 참여한 사람들중 2/3이상의 인원의 평가결과가 동일한 경우를 표시하며, 그렇지 못한 경우에는 상위 2개의 선택을 차례대로 표시한다. 상위 2개 선택의 합이 2/3미만이면 토론후 재평가를 하였다.

평가척도에서 사용하는 5단계 평가, '아주작음', '작음', '보통', '많음', '아주 많음'의 비를 '1:2:3:4:5'로 고려하는 것은 '아주많음'의 의미를 나타내기에 부족해보이므로 '1:2:4:8:16'(전 단계x2)의 비율을 사용하였다.('1:3:9:27:81'처럼 '전단계x3'으로 다음 단계를 표시하면 '아주많음'의 의 의미가 '보통'에 비해 너무 큰 것으로 평가됨.)

2.2 기존 도구 평가

2.2.1 UML(Unified Markedup Language) 평가

UML[9,10,11]은 Object Management Group의 소프트웨어 시스템 개발을 위한 Object Oriented Modeling language 표준으로 제정되어 압도적인 마케팅적인 성공을 거두었다. UML은 분석단계에서 개발자와 사용자가 구현할 시스템에 대해 정확히 이해하고 구현할 시스템의 기능을 정의하기위해 개발자와 사용자들 간에 이해, 정의된 시스템은 의사소통 과정에서의 모호함을 피할 수 있도록 검증가능한 구체적인 표현 방법으로 UML에서는 use-case diagram(Fig. 1)과 sequence diagram(Fig. 2)을 이용한다. 그리고 서로 다른 관점을 설명하기위한 7가지의 그림형식을 지원한다. 요구분석 결과를 바탕으로 실제 구현을 위해 사용될 기술과 각종 제한조건을 고려하여 즉시 프로그래밍 가능한 구체적인 해법을 제시하는 설계과정을 거치면서 구현할 시스템의 구조와 각종 클래스들의 세부사항이 결정되고 구현을 준비하게 된다.

동일한 용도의 프로그램을 아무런 도구를 사용하지 않고 절차지향언어로 개발할 경우 Ec(0/0, 4(5))에 비해, UML을 활용하여 객체지향 언어로 개발할 경우 Ec(3/3, 3)에는 비록 시스템을 설계하는 과정에서 다소의 시간이 소요되겠지만 프로그램 구현 작업이 효율적일 뿐 아니라 완성된 프로그램도 보다 체계적이고, 관리가 용이할 것이므로 설계과정의 소요시간을 충분히 상쇄할 수 있을 것이다. 단 UML은 객체를 기반으로 시스템을 설계하는 개발도구이면서 객체의 세부적인 내용에 대한 정의를 요구하므로 초보 프로그래머에게 적용되기에는 한계가 있을 것으로 판단된다.

2.2.2 OPM(Object Project Methodology) 평가

OPM[12,13,14] 또한 사람들이 개발하고자하는 시스템을 잘 이해하도록 도와서 더 빠르고 신뢰성있는 시스템을 구현할 수 있도록 지원하기위해 개발되었다. Dov Dori[12]가 정의한 OPM의 유용성[15]과 같이 OPM은 구성요소와 연관성을 나타내는 도구를 이용하여 시스템 설계자가 시스템의 세부동작을 묘사하는 OPD(Object-Process Diagram)를 작성하여 구현할 시스템 동작을 domain expert와 검증할 수 있고, 검증된 OPD는 systematica(Sight Code, 2001) 툴을 이용하여 OPL (Object-Process Language)문단(자연어지문)으로 변환되고, 변환된 텍스트 기반의 OPL문단을 통해서 설계된 OPD의 동작을 다시 검증할 수 있도록 지원한다.

동일한 용도의 프로그램을 아무런 도구를 사용하지 않고 개발할 경우 Ec(0/0, 4(5))에 비해 OPM을 활용하여 개발할 경우 Ec(4/4(3), 2)에는 비록 시스템을 설계하고 검증 과정에서 상당한 시간이 소요되겠지만 설계작업이 종료된 후의 코딩 작업은 수월하게 진행될 수 있을 것으로 평가된다. OPM의 가장 큰 문제점은 프로그램 초보자가 OPD의 기본요소간의

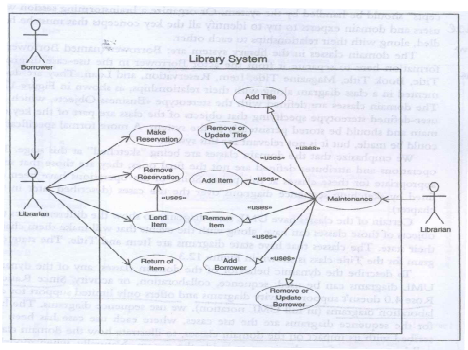

Fig. 1. user-case diagram.

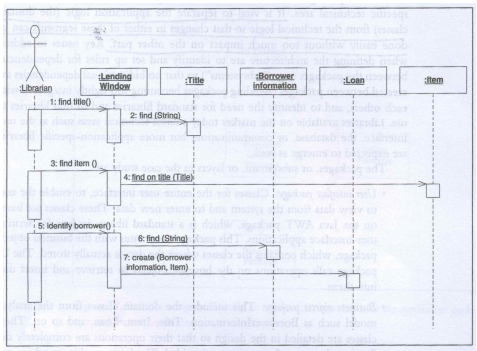

Fig. 2. sequence diagram.

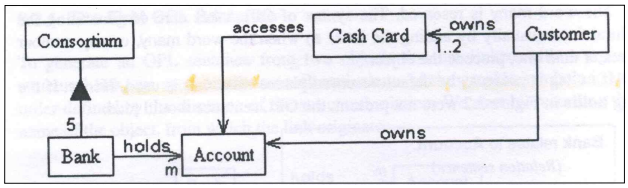

Fig. 3. OPD(object process diagram) example.

Consortium consits of **5 Banks**.
Bank holds many **Accounts**.
Customer owns **Accounts**.
Customer owns **1** to **2 Cash Cards**.
Cash Card accesses **Account**.

Fig. 4. OPL of Fig. 3.

상호작용을 이해하고 시스템의 동작을 정확하게 OPD로 그리는 것은 매우 어렵다는 점이다. 그러나 이런 어려움에도 불구하고 OPL문단은 코딩 작업시에 가장 어려운 과정인 작업순서(알고리즘)와 거의 유사하므로 OPD 작성없이 OPL문단과 유사한 정보를 체계적이고 검증가능한 과정을 통해 생성할 수 있는 방법을 제시할 수 있다면 코딩작업에 유용할 것이라 판단된다.

2.2.3 flowchart 평가

흐름도(flowchart)는 80년초반 Fortran을 배우던 시절부터 프로그램의 순서를 표현하기 위해 사용했었고, 지금도 조건문등의 실행순서를 설명할 때 사용되고 있다. 동일한 용도의 프로그램을 아무런 도구를 사용하지 않고 절차지향언어로 개발할 경우 Ec(0/0, 4(5))에 비해, 흐름도를 활용하여 절차지향언어로 개발할 경우 Ec(0/4(5), 1)에는 시스템에 적합한 흐름도를 작성하는 과정이 수행할 작업의 전부에 해당할 것으로 보여지며, 흐름도가 정확하다면 특정 언어로 코딩하는 작업시간은 해당 언어의 이해도에 따라 좌우될 것으로 평가된다. 즉 흐름도가 정확하고 개발언어에 숙달된 경우라면 단지 해당 프로그래밍 언어로 번역하는 시간정도면 충분할 것이라는 것이 공통된 평가이다.

흐름도는 배우기는 쉬우나 코딩이 가능한 수준까지 작성할 프로그램에 대해 이해를 마친 수준에 도달해야만 흐름도 작성이 가능하다는 문제점을 가지고 있다. 따라서 흐름도는 프로그램을 분석하는 과정에 대한 체계적인 방법을 제공해야 하는 프로그램 개발 보조도구로 사용하기에는 적합치 않다.

2.3 개발도구가 갖추어야 할 핵심 요소

평가결과 및 발생 가능한 평가결과의 예를 통해서 "체계적인 프로그래밍 작업 수행 지원"에 적합한 개발 도구가 갖추어야 할 조건에 대한 고려해보고자 한다. 평가결과가 Ec(5/5,5)와 Ec(0/5,5)인 경우를 상상할 수 있는데 이는 도구를 사용한 검증작업도 프로그램을 완성하는 것에 못지않게 어려우면서도 도구를 이용하여 실제 코드를 완성하는 작업량도 '아주 많음'에 해당하는 평가를 배제할 수 있어야 할 것이다. 이처럼 도구에 대한 평가결과가 최악의 평가를 획득하는 것을 방지하려면 도구를 이용한 '검증작업이 구체적인 방법과 절차에 의해 수행되고, 검증작업이 구체적인 근거를 기반으로 종료될 수 있어야 한다'는 점이 도구가 갖추어야 할 주요 요건이라 판단되며, OPM과 UML은의 평가결과는 이들이 이러한 요소를 갖추고 있음을 보여준다.

UML과 OPM의 평가에서 본 바와 같이 프로그래밍 보조 도구는 "주어진 작업을 완성하기 위해서 수행해야할 작업들이 도구상에서 구체적으로 제시되어 시각적으로 확인 가능하여야 하고, 제시된 작업들이 순서대로 모두 처리되면 목표하는 작업을 완료"할 수 있음을 검증할 수 있어야 한다. 여기서 "작업을 완료할 수 있다."는 평가는 도구 사용자의 주관적인 평가이므로 검증작업 과정이 종료되었다고 해서 해당 작업이 올바르게 수행될 수 있다는 것을 보증하는 것은 아님을 명심해야 한다. 검증과정 종료 후 코딩과정에서 검증결과의 오류가 발견될 수도 있고, 오류가 발견되면 검증과정을 재 수행하여 발견된 오류를 수정하여야 하고 생성된 코딩도 당연히 수정되어야 할 것이다. "순서대로 수행되어야할 연관된 (분할) 작업들"에서 분할 작업이란 각 작업들을 독립적으로 처리할 수 있음을 의미하고, '순서대로' 연관되어 있다는 의미가 더해지면 각 작업이 독립적으로 차례대로 완성되면 자동적으로 전체작업이 완성됨을 의미한다. 순서를 가지는 분할된 작업은 단계별 검증을 지원할 뿐 아니라 작업 분할이 가시적으로 이루어지면 분할의 타당성도 검증할 수 있게 된다.

UML과 OPM과 달리 사용의 편의성을 위해 그림을 사용하지 않고 "수행해야할 작업들이 순서대로,

모두 제시되었음"을 "체계적으로 검증"할 수 있으려면 "수행할 큰 작업이 순서대로 수행되어야 할 연관된 작업들로의 재귀적분할(divide and conquor, top-down 접근)을 통한 단계적 검증"을 지원하는 것이 완성된 수행작업목록에서 오류발생가능성을 줄일 수 있는 방법일 것으로 판단된다.

3. 개발 보조도구 제안

3.1 기존 보조도구의 문제점

개발도구가 갖추어야할 핵심 요소로는 개발도구는 수행할 작업을 처리할 방법을 체계적으로 검증할 수 있어야 한다는 것을 들 수 있었는데 체계적인 검증이란 "수행할 작업이 서로 연관되며 순서대로 수행할 수 있는 작업들로 재귀적으로 분할되어야 하며, 이러한 과정이 시각적으로 검증될 수 있도록 지원하는 것"을 의미한다. 이러한 핵심요소를 고려하여 각 개발도구를 평가한 결과에 따라 도출된 문제점을 재정리하면 다음과 같다.

UML은 객체를 기반으로 시스템을 설계하는 보조도구로써는 핵심요소를 지원하는 것으로 평가되지만 UML의 개발도구로써의 이점이 초보 프로그래머에게 적용되기에는 부적절한 것으로 판단된다. OPM 또한 보조도구로써는 핵심요소를 지원하며, OPL문단은 효율적인 코딩작업에 큰 도움이 되겠지만 이를 위해서 시스템의 동작을 정확하게 OPD로 그리는 것은 초보프로그래머에게는 불가능해보인다는 점이다. 흐름도는 배우기는 쉽지만 프로그램에 대한 이해가 코딩수준까지 이루어진 뒤에야 흐름도를 사용할 수 있다는 점에서 프로그램 개발도구로써의 역할은 미미해보인다.

3.2 개발 보조도구로써 Outlining tool 적절성 평가

프로그래밍 개발 보조도구는 초보 프로그래머도 쉽게 사용할 수 있는 체계적인 검증 방법을 제공할 수 있어야 한다. 이를 위한 프로그램 개발 보조도구가 UML과 OPM처럼 특별한 인터페이스를 가질 수 있지만 복잡한 인터페이스와 개념은 초보 프로그래머에게는 장애가 될 수 있다. 그러므로 OPD를 통하지 않고도 OPD 그리기 과정에서 수행되는 체계적인 검증과정의 핵심요소인 "가시적인 검증"을 제공하고 "순서를 가지는 작업들로 재귀적 분할"을 통해 OPL문단 수준의 문장들을 생성할 수 있어야 하고, "초보프로그래머가 사용하기 수월하여야 한다."는 점이 본 연구에서 추구하는 프로그램 개발 보조도구가 갖추어야 할 핵심 기능이라 요약할 수 있다. 이러한 기능을 갖춘 프로그램 개발 보조도구를 상상해보자.

재귀적분할과 가시적인 검증 과정을 지원할 수 있고, 사용하기 쉬운 도구는 굳이 응용프로그램을 개발하지 않더라도 이미 생활속에서 사용되는 도구들 중에서도 발견할 수 있는데 그 예로 편집기(흔글, MS Word)의 표와 개요(목록)기능을 들 수 있다. 편집기(흔글, MS Word)의 표 기능을 활용하여 수준별 작업을 칸으로 구분하여 표시하고, 각 수준에서 재귀적으로 분할된 세부작업들은 새로운 칸에 분할된 작업의 수만큼 열을 확장하여 표시(표의 칸, 열 추가기능)할 수도 있다. 그러나 표를 편집, 관리하는 기능이 복잡할 뿐 아니라 작업이 커지더라도 항상 전체 작업을 동시에 편집하는 등 수준별 작업관리가 불가하여 관리하기 불편해진다. 아래의 Fig. 5처럼 수준A와 수준B로 구성된 작업에서 '수준A'가 다시 4개의 작업으로 분할되고 순서를 표시할 수 있으면, 동일한 표편집 과정을 통해서 재귀적인 분할을 나타낼 수 있고, 시각적인 검증도 가능하나 표편집과정(칸, 줄 나누기/추가)이 번거롭고, 전체표가 늘 펼쳐진 상태로 표시되므로 수준별 작업 파악이 어렵다. 아래 Fig.5의 왼쪽 표에서 칸과 열이 추가되어 오른쪽 표로 바뀌면 왼쪽 표로 전환될 수 없다.

흔글의 개요(목록)기능(MS word의 목록기능도 유사함)을 활용하면 위의 표로 작성된 작업목록을 Fig. 6과 같이 나타낼 수 있지만 현재 흔글의 개요(목록)기능에는 목록 닫기/펼치기 기능을 지원하지 않으므로 수준이 추가, 확장되면 이전으로 돌아갈 수 없어 Fig. 5의 표를 사용한 경우처럼 전체 개요를 한꺼번에 참조할 수 밖에 없다는 단점은 여전하다.

작업A
작업B

가능 → ↚ 불가능

작업A	
	작업A.1
	작업A.1
	작업A.3
	작업A.4
작업B	

Fig. 5. Recursive division using HanGul table.

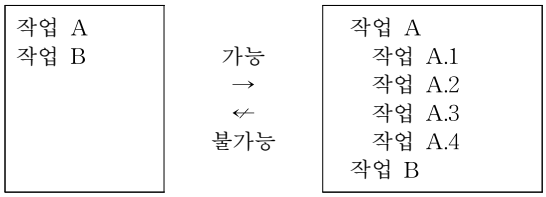

Fig. 6. recursive division using HanGul outling function.

MS Word나 흔글의 목록 기능의 문제점인 수준별 작업관리를 위해서 정통적인 outlining 도구를 활용할 수 있다. Outlining 도구중에서 흔히 접할 수 있는 Microsoft Word 2010의 개요기능이나 개요도구(예, FreeMind, 무료)를 활용하면 수준표시/감추기 기능을 활용하여 수준별 작업관리가 가능하다. 그리고 C/C++ 프로그래밍도구인 Visual Studio 2010 개발도구에서 프로젝트 개발시에 MS word 편집 파일인 .docx 파일과 FreeMind의 편집 파일인 .nm 파일을 솔루션에 추가시켜 Fig. 7처럼 소스프로그램 개발과 병행할 수도 있다.

단 이 경우는 Visual Studio 2010 개발도구의 편집창에서 직접 MS word (FreeMind) 파일을 편집할 수 없고, MS word나 FreeMind를 별도창에서 열어서 해당 파일을 편집하게 된다. 또한 docs/mm 파일을 솔루션에 추가하려면 외부에서 docs/mm 파일을 먼저 생성한 후에 솔루션 이름을 우클릭한 후, "추가->기존항목->편집된 파일 선택"과정을 거쳐 편집된 '개요작성.docx (free1.mm)'파일을 솔루션에 추가할 수 있다. 분리된 환경이 불편하다면 개요기능만을 포함하는 편집기를 만들어서 Visual Studio 2010 개발도구의 편집창으로 사용할 수 있게 통합환경을 구축하는 방법에 대한 연구가 필요할 것으로 판단된다.

Fig. 7. Including external files.

3.3 Outlining Tool인 MS-Word 개요기능을 사용한 사례

본 논문에서는 개요도구이며, 쉽게 접할 수 있는 Microsoft word 2010의 개요기능이 어떻게 프로그래밍 보조도구로 활용될 수 있는지를 실례를 통해서 보이고자 한다. 프로그래밍 작업이란 "수행할 작업을 재귀적으로 세부작업으로 적절히 분할[16]하면서 분할된 세부작업들의 수행 순서를 정하고, (세부)작업의 재귀적 분할이 종료된 후, 해당 작업들을 프로그래밍 언어로 번역(코딩)하는 작업"을 말한다. 이때 작업의 분할은 사용할 프로그래밍 언어로 직접 번역 가능한 작업단위와 동일할 때까지 반복하여야 하나 프로그래밍 언어에서 표현할 수 있는 작은 작업단위의 크기는 해당 프로그래밍언어의 숙련도에 따라 좌우될 수 있다.

3.3.1 프로그램 적용 사례 1

적용사례를 알아보기위해 가장 간단한 프로그램을 생각해본다. 즉 "두 수를 입력하여 합을 출력한다." 이러한 요구를 만족하는 프로그래밍 작업을 위해서 (사람수준에서)수행할 일의 절차를 따져보자. 우선 1)두개의 수를 입력하고, 2)두 수의 합을 구하고, 3)결과를 출력하는 3개의 작업으로 분할하고 이를 MS word에서 아래와 같이 작성하고 저장한 후, "개요작성연습1" 파일을 솔루션에 추가한다.

Fig. 8의 "개요작성연습1.docx" 파일을 프로그램 숙련자가 본다면 코딩 작업을 시작할 수 있을 정도로 상세하지만 초보자라면 프로그램 문법과 코딩작업을 고려하여 문장(작업)을 프로그램에 적합하도록 세분화하여야 할 것이다. 각 문장들이 세분화된 결과는 Fg. 9와 같다.

Fig. 9와 같이 분할된 세부 작업들은 프로그래밍

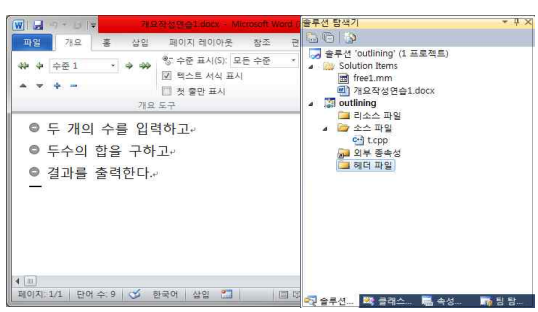

Fig. 8. Start of Programming, make job list.

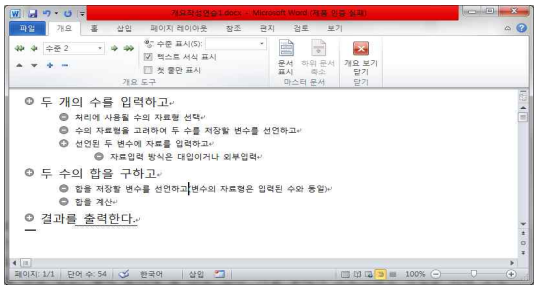

Fig. 9. Programming, Job Division.

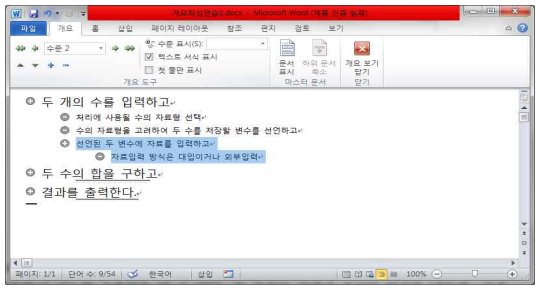

Fig. 10. fold/unfold job list,

언어에 대한 기본적인 이해만 있다면 충분히 코딩(번역)이 가능한 수준임을 알 수 있다. Fig. 9, Fig. 10의 각 문장들의 앞에서 표시된 원기호 내부에 +기호가 포함되어 있거나 -가 포함되어 있는데 +가 포함되어 있으면 하위 세부작업 목록이 있음을 나타내므로, 해당 기호를 클릭하여 Fig. 10처럼 “두수의 합을 구하고”단계의 세부작업을 펼치거나 가릴 수도 있다. 이를 통해서 특정 수준별로 수행할 작업을 검증할 수 있다.

3.3.2 프로그램 적용 사례 2

적용 사례 1에서는 단순히 개요도구를 사용하여 프로그램작업을 수행하는 방법과 프로그램 작업결과 개요도구에 표시되는 정보와 코딩의 의미를 알 수 있었다. 이제는 각 작업이 ‘재귀적으로 분할처리

두 정수(>0)의 최대공약수, 최소공배수 산출 소스프로그램 작성 풀이과정)

1)

```
2 | 64  40
2 | 32  20
2 | 16  10
      8   5
```

최대공약수 = 8 = (2*2*2)
최소공배수 = 320 = (8*8*5)

2)

```
3 | 60  45
5 | 20  15
      4   3
```

최대공약수 = 15 = (3*5)
최소공배수 = 180 = (15*4*3)

제약조건)

1) **풀이과정을 반드시 출력해야 함.**

1)

```
2 >> 64  40
2 >> 32  20
2 >> 16  10
       8   5
```

2)

```
3 >> 60  45
5 >> 20  15
       4   3
```

2) **출력메시지**

1. 00와 00의 최대공약수는 00, 최소공배수는 00입니다.
2. 00와 00는 서로소입니다.

된다.’는 의미를 좀 더 알아보고자 한다. 이번에 좀더 복잡한 프로그래밍 작업을 고려해보려한다. 아래 상자속의 문제에 대한 설명을 참조해보자. 주어진 문제를 풀이하는 프로그래밍 작업을 위해서는 우선 문제에 대한 이해를 위해 우선 종이와 연필을 꺼내서 2가지 입력형태 1) 64, 40 2) 60, 45에 대해 최대공약수, 최소공배수를 계산해보자.

풀이를 통해 우선 최대공약수를 구하고, 이후에 서로소가 아니면 최대공약수를 이용하여 최소공배수를 계산할 수 있음을 알 수 있다. 프로그램에서는 최대공약수를 구하는 풀이과정을 출력하고, 풀이과정이 끝나면 최대공약수와 최소공배수를 출력한다. 물론 최대공약수가 1인 경우는 두수는 서로소임을 출력한다. 풀이과정을 보여주는 출력과정이 손으로 풀이하는 과정과 조금 차이가 있음을 알 수 있다. 예를 들어 첫 번째 예제의 경우 우리는 64, 40을 쓴 후, 약수를 계산하고 약수가 2라고 판정되면 2라는 수를 64앞에 쓰게 된다. 그러나 콘솔창에 결과를 출력할때는 40을 출력한 후 커서를 64앞으로 이동할 수 없다. 따라서 64와 40의 약수인 2를 찾은 후에 찾은 약수를 출력하고 ‘>>’표시를 출력하고, 그리고 64와 40을 출력하여야 한다. 다음 단계에서는 32와 20의 약수를 찾은뒤 약수와 함께 출력된다. 그리고 ‘서로소’인 경우에 대한 연습을 위해 또 다른 입력 3, 7을 처리해본다.

분석과정이 종료되면 코딩작업을 시작하는게 통상적이지만 분석결과를 사용하여 MS word의 개요기능으로 구체적으로 수행할 세부 작업목록을 작성하면서 분석결과를 검증해보자. Fig. 11은 개요기능을 활용하여 시작단계에서 수행할 작업목록을 설명하고 있다. 아래와 같이 차례대로 Fig. 15까지 완성한 후 코딩 작업을 시작해보자. 이러한 과정을 거쳐 작성된 프로그램의 평가는 Ec(0, 4(3))을 Ec(1/3, 2(1))

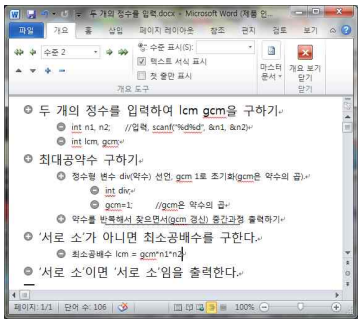

Fig.11. Top Level expansion.

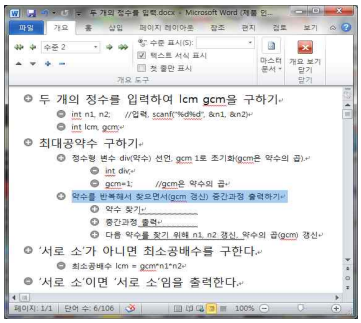

Fig.12. Top+1 Level expansion.

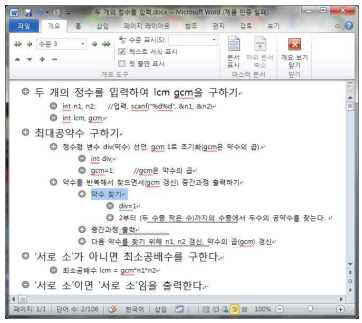

Fig.13. Top+2 Level expansion.

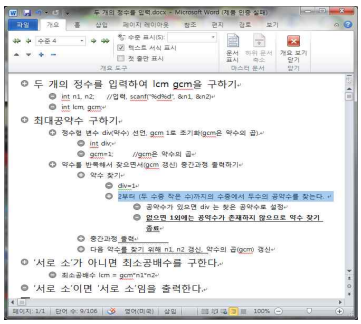

Fig. 14. Top+3 Level expansion

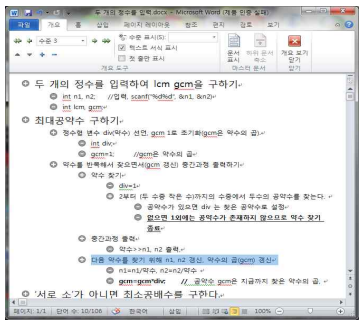

Fig. 15. Top+2 Level expansion(cont.).

으로 개선할 수 있다.

3.3.3 프로그램 적용 사례를 통한 고찰

초보자를 위한 C프로그래밍실습강좌에서 기본문법(변수,조건문, 반복문)에 대한 강좌가 진행된 후, 동일한 시간에 단순히 문제를 말로써 설명한 경우에 비해 개요도구를 활용하여 문제를 설명한 경우의 효과를 비교하는 실험을 수행하였다. 실험군은 학번에 따라 임의 분류된 4개의 실험분반이며, 적용사례2에서 제시한 문제를 사용하였다. 2개의 실험그룹(A, B)은 각 2분반으로 구성된다. 평가는 문제를 구두로 설명하는 것으로 시작되고, 개요도구의 효과를 정확히 평가하기위해 30분이내에 해당 문제를 해결한 학생은 평가대상에서 제외하였다. 30분이 지난 후 A그룹에서는 해당 문제를 다시 동일한 방식으로 설명을 하고, B그룹은 개요도구로 작성된 문제분석 자료를 제시하면서 설명하였다. 이후 30분씩의 두 번에 걸친 실습시간 동안 평가를 수행하였다. 평가 후에는 프로그램을 완성하지 못한 학생을 대상으로 설문조사를 수행하여 해당 문제에 대한 이해도를 평가하였다. 이때 사용한 설문문항은 다음과 같다. 1) 문제풀이 방법은 충분히 이해하였으나 문법활용능력이 부족하여 프로그램이 완성되지 않았다. 2) 문제 풀이방법에 대한 이해가 부족하여 프로그램이 완성되지 못하였다.

Table 1. Result of Experiment

Group(Member) \ time (minute)		frist 30 min.	second 30 minute			third 30 minute			persons who completes after first 30	comprehension evaluation (select 1/reaminder)
			10	20	30	10	20	30		
Group A (repeat explanation)	1(18)	3		1	1		1		3	3/12(25%)
	2(22)	4	1		1	1		1	4	4/14(28%)
Group B (explain with Outlining tool)	1(21)	3		1	2	1	1		5	7/13(53%)
	2(24)	3		2	1		2	1	6	8/16(50%)

평가결과 및 설문결과는 Table 1과 같다. Table 1에 따르면 평가시간이 지날수록 개요도구를 이용하여 설명을 제공한 그룹에서 작으나마 프로그램을 완성한 수가 많지만 실험군이 작아 개요도구를 사용함에 따른 이점으로 평가하기에는 부족해보이지만, 평가 후 학생들 스스로가 문제에 대한 이해도를 평가한 결과는 개요도구의 사용이 유용함으로 보여주는 의미있는 결과라 판단된다. 결국 일부 학생들이 비록 프로그램 문법을 활용하는 능력이 부족하여 오류없는 프로그램을 완성하지는 못했지만 스스로 문제에 대한 이해를 제대로 하고 있다고 스스로 평가하는 것을 고려하면 시간이 지나면 완성된 결과를 도출할 가능성이 훨씬 높다는 것을 보여주고 있다고 할 것이다. 결국 설계도구로써 개요도구의 유용성의 평가를 "주어진 문제를 분석하여 문제를 바르게 이해하는데 도움을 주는가?"라고 질의한 결과를 놓고 판단한다면 개요도구는 설계도구로써 유용하다는 평가를 받기에 부족함이 없으리라 판단된다.

또한 개요도구를 이용한 문제 설명시에는 단계별로 프로그램 완성자가 발생하는 것을 볼 수 있었는데 이는 학생들의 프로그래밍 능력에 따라서 준비해야 할 작업 목록에서의 작업의 크기가 다름을 보여주는 흥미로운 결과라 할 수 있다. 이는 작성할 작업에 대한 설명이 학생들의 수준에 적합한 눈높이 설명이 가능하다는 것을 보여고 있다고 판단된다. 학생들도 스스로 개요도구의 활용한다면 직접 코딩이 가능한 수준까지 작업분석을 시도함으로써 효과적으로 프로그래밍 작업을 완료할 수 있을 것으로 판단된다. 실험을 통해 문제를 단순히 해결할 수 있을만큼 이해한다는 것과 코딩수준의 이해에는 갭(간격)이 존재하는 것을 알 수 있었고 많은 초보 프로그래머들이 그 갭을 극복하여 프로그래밍의 세계로 진입하는데 개요도구가 큰 도움이 될 것이라 판단된다. 또한 작성된 개요를 통해 소스프로그램이 순서대로 처리되어야 하는 작은 작업들의 모임이라는 중요한 의미도 이해할 기회를 제공한 것으로 판단된다.

4. 결 론

본 연구에서는 초보 프로그래머를 위한 프로그램 보조도구를 제시하였다. 이를 우선 기존의 프로그램 설계도구들이 프로그램 설계 보조도구로써의 역할을 충실하게 수행하고 있는지 여부를 평가하기 위해 평가척도를 제시하고, 기존 보조도구들을 평가하였다. 평가결과를 바탕으로 본 연구에서는 프로그래밍 과정을 첫째, 문제분석(이해)단계, 둘째, 프로그램 작성과정(절차수립, 모의실행 및 검증 과정) 즉 수행할 작업을 처리할 단계별 세부작업목록들로 변환하여 '코딩이 가능한 상세한 수준'으로 변환하는 과정, 그리고 마지막으로 코딩과정으로 분할이 필요함을 제시하였다. 그리고 기존의 프로그래밍 보조도구인 UML과 OPM, 그리고 흐름도의 문제점을 분석을 통해 초보프로그래머가 쉽게 활용할 수 있는 프로그램 개발 보조도구를 위해 OPL과 유사한 수준의 문장들을 생성하는 보조도구로 개요도구를 사용할 것을 제안하였다.

본 연구에서 제안한 개요도구를 활용한 프로그램 작성이 성공적일수록 코딩작업 또한 더 쉽게 진행될 수 있음을 적용 사례를 통해 확인할 수 있었다. 그러나 소스프로그램 개발도구로 사용하는 Visual Studio 2010과 개요도구인 MS word가 별도로 사용하는데 따른 불편함을 해소할 수 있도록 Visual Studio와 통합하여 사용할 수 있는 개발환경을 구축하는 것에 대한 연구가 필요해보인다. 통합환경이 구축되면 개요편집 파일을 프로젝트의 소스파일처럼 직접 추가 및 편집이 가능하여 보다 편리하게 사용할 수 있을 것으로 기대된다.

REFERENCE

[1] Y.G. Kim and H.S. Han, "A Study on the Possibility of Teaching Computer Programming using Spreadsheets," *Journal of Education & Science Research Institute*, Vol. 42, No. 2, pp. 191-209, 2011.

[2] S.J. An and Y.J. Lee, "A Study of Programming Attitude," *Proceeding of The Korean Assocication of Computer Education,* Vol. 18, No. 2, pp. 21-24, 2014.

[3] W.Y. Cho, "How to Teach Algorithm?," *Journal of Korea Society of Mathematical Education Series A: The Mathematical Education*, Vol. 39, No. 1, pp. 49-58, 2000.

[4] J.P. Cho and Y.J. Lee, "The Development of

Learning Program using Scratch to Foster Logical Thinking Ability of Middle School Students in Technology Education," *Korean Journal of Technology Education Association*, Vol. 12, No. 1, pp. 213-233, 2012.

[5] J.E. Shim, J.Y. Go and J.C. Shim,"A Study on Training Courses Development and Analysis for Improving the CReativity using Arduino," *Journal of Korea Multimedia Society*, Vol. 17, No. 4, pp. 514-525, April 2014.

[6] C.G. Ryu and C.H. Lee, "Effects that Scratch Programming has on Creative Problem-solving for Gifted Elementary Students," *Journal of Korean Practical Arts Education*, Vol. 25, No. 1, pp. 149-169, 2011

[7] M. Kim and T. Lee, "Development of the Software Education Program using LEGO WEDO," *Proceeding of The Korean Association of Computer Education, Vol.* 18, No. 2, pp. 37-40, 2014.

[8] L.J. Morell, "Algorithms First," *Proceeding of Consortium for Computing Science in Colleges Mid-south Conference*, pp. 48-55, 2005.

[9] H.E. Eriksson and M. Pneker, *UML Toolkit*, John Wiley & Sons, NY, USA, 1997.

[10] E. Arisholm, L.C. Briand, S.E. Hove, and Y. Labiche, "The Impact of UML Documnetation on Software Maintenance: An Experimental Evaluation," *IEEE Transactions on Software Engineering*, Vol. 32, No. 6, pp. 365-381, 2006.

[11] B. Anda, K. Hansen, I. Gullesen, and H.K. Thorsen, "Experience from Introducing UML-based Development in a Large Safety-critical Project," *Empirical Software Engineering*, Vol. 11, No. 4, pp. 555-581, 2006.

[12] I. Reinharts-Berger and D. Dori, "Object-Process Methodology(OPM) vs. UML : A Code Generation Perspective," *Proceeding of The 16th Conference on Advanced Information Systems Engineering, Riga, Latvia, Knowledge and Model Driven Information Systems Engineering for Networked Organisations,* Vol. 1, pp. 275-286, 2004.

[13] D. Dori, *Object-Process Methodology-A Holistic Systems Paradigm*, Springer Verlag, New York, 2002.

[14] I. Reinharts-Berger and D. Dori, "OPM vs. UML - Experimenting with Comprehension and Construction of Web Application Models," *Empirical Software Engineering*, Vol. 10, No. 1, pp. 57-79, 2005.

[15] Patricia Resende, http://www.bizjournals.com/oldmht/stories/200/01/ 15/story2.html?pag=all (accessed Aug., 25, 2015).

[16] G.A. Miller, "The Magical Number Seven, Plus or Minus Two: Some Limits on Our Capacity for Processing Information," *Journal of The Psychological Review,* Vol. 63, No.2, pp. 81-97, 1956.

박 세 명

1985년 2월 경북대학교 전자공학과 전산공학 석사
1994년 8월 경북대학교 전자공학과 전산공학 박사
1990년 3월~현재 인제대학교 컴퓨터공학부 교수

2003년~2004년 인제대학교 정보처장
2001년~University of Minnesota, 2009 University of Washington 교환교수 한국정보과학회, 한국멀티미디어학회 종신회원
관심분야 : 빅데이터 분산컴퓨팅, 네트워크보안 및 침입탐지, 프로그래밍 교육방법론