예제가 프로그래밍 언어의 학습과정에 미치는 영향*

The Impacts of Examples On the Learning Process of Programming Languages

김 진 수** 김 진 우**
(Jin-soo Kim) (Jin-woo Kim)

요 약 예제에 의한 학습은 프로그래밍 언어를 포함한 다양한 주제들을 숙지하는데 효과적인 방법으로 밝혀져 왔다. 그러나 어떤 예제를 어떻게 제공하는 것이 바람직한가에 대한 보다 심층적인 연구는 많지 않다. 본 연구는 예제가 제시되는 방식과 제시되는 예제의 형태가 예제에 의한 프로그래밍 언어의 학습 성과에 영향을 미치는 두 가지 중요한 차원이라는 가설을 세웠다. 이 가설들을 자바 프로그래밍 언어의 학습 과정을 통하여 검증하기 위하여 컴퓨터 상에서 실험을 실시하였다. 예제의 제시 방식에서는 두 종류의 예제들을 부가적 설명 없이 제공하는 것이 부가적 설명과 함께 하나의 예제를 제공하는 것보다 더 효과적이라는 결과를 얻었다. 예제의 형태에서는 두 종류의 예제를 제공받았더라도 두 예제가 주어진 과제와 기능적으로 유사한 경우가 기능적으로 상이한 경우보다 더 나은 수행 결과를 나타냈다. 이와 같은 수행 결과의 차이에 대한 이유를 밝히기 위해 개별 피험자들의 프로그래밍 행동의 유형을 시간과 빈도의 관점에서 분석하였으며 또한 피험자들의 행위에 대한 보다 체계적인 설명을 위하여 GOMS 모델을 제시하였다. 결론적으로, 본 연구의 결과들은 프로그래밍 언어를 효과적으로 지도할 수 있는 교육 시스템 개발에 기여할 수 있을 것으로 기대된다.

주제어 예제 중심 프로그래밍, 컴퓨터 언어 학습, 프로그램 유사성

Abstract Learning by examples has proven to be an efficient method in mastering various subjects including programming languages. This study hypothesizes that the number of examples and the type of examples are two significant dimensions that influence the performance of learning programming languages by examples. A set of experiments was conducted to investigate the impacts of the two dimensions in the domain of JAVA programming. The results showed that providing two examples is more effective than providing only one example even though significantly more explanations are attached to the single example. Among the 'two-example' groups, the group that was given functionally similar examples performed better than those with functionally dissimilar examples. Explanations for these results are provided in this paper based on the behavioral patterns of individual subjects in terms of time and frequency. This paper concludes with the implications of the study results for the development of effective tutoring systems for programming languages.

Keywords Example-based programming, computer language learning, program similarity

1. 서 론

많은 사람들이 인터넷을 손쉽게 사용할 수 있게 됨으로써 인터넷을 통한 정보의 교류가 활발해 지고 있으며 이에 따라 정보의 중요성이 그 어느 때보다도 크게 부각되고 있다. 인터넷을 통해 전달되는 정보들

* 본 연구는 (1998)년 학술진흥재단의 학술 연구비에 의하여 지원되었음. 본 연구에 대한 문의는 서울특별시 서대문구 신촌동 연세대학교 상경대학 경영학과 611호 김진우 교수 (jinwoo@base.yonsei.ac.kr)로 해주시기 바랍니다.

** 연세대학교 인지과학 협동과정

은 대부분 컴퓨터에 의해서 디지털 형태로 제작되거나 가공되고 있다. 이러한 정보의 생성과 가공 과정을 지원하기 위해서는 더욱더 많은 컴퓨터 소프트웨어와 하드웨어 시스템들이 요구되고 있다. 하드웨어 시스템의 경우에는 일단 개발이 되면 동일한 제품을 반복적으로 생산할 수 있지만 소프트웨어의 경우에는 응용 분야에 따른 다양한 요구가 존재하기 때문에 보다 많은 컴퓨터 프로그래머들을 필요로 하게 된다.

이처럼 컴퓨터 프로그래머에 대한 수요가 증가하고 있음에도 불구하고 컴퓨터 프로그래머를 교육시키는 방법은 여전히 낙후되어 있다. 전통적인 프로그래밍 교육 방법은 주로 컴퓨터 언어의 구성 요소들을 설명하고 그에 대한 간단한 프로그램 코드들을 보여주는 것이다. 이러한 방법은 프로그래밍 언어를 학습하는 학습자들(이하 프로그램 학습자)에게 인지적 부담을 줌으로써 컴퓨터 프로그래밍 언어에 대한 학습자들의 학습 동기를 위축시킨다. 즉, 전통적인 훈련 방법은 프로그램 학습자들에게 프로그래밍 언어의 형식과 의미를 기억하도록 강요한다. 언어의 형식은 컴퓨터 프로그램을 작성하는데 적용되는 규칙을 의미하고 언어의 의미는 프로그래밍 언어의 구성 요소가 가지는 의미를 말한다[2]. 예를 들어, add(button);이라는 문장을 보면 이 문장의 의미는 버튼을 화면에 보여준다는 것이고 그 규칙은 괄호 안에 화면에서 보여줄 대상을 넣고 문장의 끝에는 세미콜론을 붙이는 것이다. 이와 같이 전통적인 학습방법은 프로그램 언어의 기초적 형식과 의미에 관련된 문제뿐만 아니라 프로그램 학습자들이 프로그램을 하고자 할 때 필요한 언어적 구성요소의 선택과 그들의 결합 방식에 대한 정보들을 거의 제공하지 못하고 있다.

이러한 전통적인 학습 방법이 가지고 있는 인지적 부담과 프로그램 과정에서 필요한 언어적 구성 요소의 선택 그리고 구성 요소들 간의 결합 방식 등과 관련된 문제들에 대한 해결책으로 제시되는 것이 바로 예제 중심 학습 방법이다 [7]. 만일 프로그램 학습자가 프로그래밍 언어를 학습하는 과정에서 예제들을 참조할 수 있다면 인지적 부담이 현저히 줄어들 것이다. 이는 프로그램 학습자들이 프로그램의 과정에서 필요한 정보가 있을 때, 그들이 기억해야 할 것은 그 정보가 예제의 어느 부분에 있는가 하는 것뿐이기 때문이다. 즉 프로그램 학습자는 그 정보의 위치를 기억에서 인출하고 그 정보가 있는 곳에 가서 필요한 정보를 참조하면 된다. 이는 프로그램 학습자가 자신의 외부에 있는 예제와 상호작용을 함으로써 자신의 지적 능력을 보다 효율적으로 사용하는 한 방법이 될 수 있다.

게다가 예제에서 주어지는 프로그램 코드들은 프로그램의 언어적 구성 요소들이 조직되는 방법에 대한 힌트를 줄 수 있다[3]. 만일 프로그램 학습자가 예제의 프로그램 코드와 프로그램 코드를 실행시켜서 얻은 실행 결과간의 관계를 이해할 할 수 있다면, 유추 과정을 통해서 다른 유사한 실행 결과에 대한 프로그램 코드를 작성할 수 있다. 물론 그와 같은 방식으로 프로그래밍을 하기 위해서는 예제의 실행 결과와 주어진 프로그래밍 과제의 실행 결과가 유사성을 가지고 있어야 한다는 전제를 필요로 한다.

그렇다면 먼저, 예제의 프로그램 코드와 프로그램 실행 결과를 어떻게 제공하는 것이 프로그램 학습자들에게 그 둘 간의 관계를 보다 용이하게 이해하도록 할 것인가 하는 것이 문제가 된다. 프로그램의 실행 결과는 화면을 통해 가시적으로 확인할 수 있기 때문에 쉽게 이해할 수 있겠지만 프로그램 코드는 그 자체의 규칙과 의미를 가지고 있으므로 프로그램 학습자가 이해하는데 곤란을 느낄 가능성이 있다. 따라서 프로그램 코드를 프로그램 학습자가 쉽게 이해하도록 제시하는 방법이 프로그램 학습자의 학습성과에도 커다란 영향을 끼칠 수 있을 것으로 예측된다.

프로그램 코드를 프로그램 학습자들이 보다 쉽게 이해하도록 제공할 수 있는 두 가지 방법이 있다. 그 하나는 전통적 프로그램 교육 방법에서와 같이 예제의 프로그램 코드에 있는 각각의 언어적 구성 요소들에 대한 언어적 설명 및 그 구성 요소가 포함된 프로그램 코드를 함께 제공하는 것이다. 이 경우에 기존의 교육 방식과의 차이는 예제가 제공됨으로써 프로그램 학습자의 인지적 부담을 줄이고 프로그램 코드의 구성 요소들 간의 결합 방식에 대한 힌트를 제공한다는 점이다.

또 다른 한 가지 방법은 두 가지 사전 유사물(prior analog)을 제공하는 것이 조직화된 지식을 표상하는 문제의 스키마를 이끌어 내는데 도움이 된다는 연구 결과에 그 기초를 두고 있다[2]. 즉, 언어적 구성 요소에 대한 설명이나 그 요소가 포함된 몇 줄의 프로그램 코드 대신에 또 다른 예제를 하나 더 제공하는 것이다. 후자의 경우에는 예제들 간의 비교 행위만을 통해서 프로그램 코드를 이해해야 하기 때문에 유추적 프로그래밍을 촉진할 것으로 예측된다. 그 반면에, 전자의 경우에는 프로그램 학습자로 하여금 주어진 프로그램 코드를 언어적 설명을 통해서 이해하고

이를 기초로 프로그램을 작성하도록 유도할 수 있다.

예제를 제공하는 방법과 더불어, 예제의 형태도 프로그래밍 언어의 학습에 중요한 역할을 할 수 있다. 앞서 지적하였듯이 예제의 실행 결과는 주어진 프로그래밍 과제의 실행 결과와 유사성을 가져야 한다. 그 유사성은 외형과 기능이라는 두 가지 형태를 취할 수 있다. 실행 결과의 외형이란 프로그램의 실행 결과가 컴퓨터 모니터 상에서 어떻게 보여지는가를 의미하고 실행 결과의 기능은 프로그램이 어떤 일을 하는가를 의미한다.

만일 하나의 예제만 제공될 경우, 그 예제의 실행 결과는 당연히 외형과 기능이 프로그래밍 과제의 실행 결과와 유사해야 한다. 그러나 두 개의 예제를 제공하는 것이 더 효과적이라고 판명된다면, 두 번째 예제로 어떤 형태의 유사성 즉, 외형적으로 유사한 예제가 더 효과적인지 아니면 기능적으로 유사한 예제가 더 효과적인지를 실험을 통해 확인해 볼 필요가 있다.

이 연구의 목적은 컴퓨터 프로그래밍 교육에 예제를 도입하는 것이 효과적이라는 전제아래서 어떤 예제를 어떻게 제공하는 것이 프로그램 학습자에게 도움이 될 것인가를 실험을 통해 검증하는 것이었다. 예제의 제공방식에 따른 프로그래밍 과제 수행의 효과를 알아보기 위하여 피험자들을 하나의 예제와 부가적 설명을 제공한 집단(이하 「예제1+ 설명」 집단)과 두 예제를 제공한 집단(이하 「예제1+예제2」 집단)으로 나누어 실험을 실시하였다. 이때 예제1과 프로그래밍 과제는 모든 집단에 동일하게 제공되었다. 한편, 「예제1+예제2」 집단에서 예제2를 외형적 유사성과 상이성 그리고 기능적 유사성과 상이성에 따라 4개의 집단으로 세분화하여 예제의 유사성의 형태에 따른 프로그래밍 학습 성과를 측정하였다.

2. 예제 중심 학습의 이론적 근거

2.1 예제 중심의 프로그래밍[1)]

기존의 많은 연구에서 예제들이 학습에 유용하다고 주장되어 왔다 [12, 6]. 예를 들어, Xinming Zhu는 학생들에게 잘 선택된 일련의 예제들을 제공하여줌으로써 다른 직접적인 교수가 없이도 수학의 다양한 기능들을 가르칠 수 있다는 사실을 밝혀냈다 [12].

예제는 또한 프로그래밍에서 다양한 방법으로 사용될 수 있다. 예를 들면, 컴퓨터 언어의 구조를 보여준다거나, 알고리즘을 구현한다거나, 문제에 대한 해결책을 제시한다거나 또는 프로그래밍 형태에 대한 예로서 활용될 수 있다[6].

프로그래머가 프로그래밍을 하는데 알아야 할 두 가지 요소들이 있다. 그 하나는 프로그래밍 언어의 형식이고 다른 하나는 프로그램 의미이다. 프로그램 코드의 의미에 대한 이해 없이는 그 코드를 유사 과제에 적용하는 것이 거의 불가능하다. 프로그램 코드의 의미뿐 아니라 형식도 프로그래밍에서 중요한 부분이다. 프로그램 코드는 주어진 형식에 맞추어 작성하지 않는다면 컴퓨터 시스템이 이를 처리할 수 없기 때문이다. 예제의 프로그램 코드를 제공하는 것은 이 두 가지 측면을 모두 만족시킬 수 있다는 점에서 장점을 가진다. 예제는 프로그램의 형식적 구조에 대한 힌트를 제공하며, 적은 정도이기는 하나 프로그램의 의미 이해에도 영향을 끼치기 때문이다[6].

그러나 예제를 제공하더라도 어떤 방식으로 제공하느냐에 따라 과제 프로그램의 수행 성과가 차이가 날 수 있다. 프로그래머들이 주어진 예제들로부터 힌트들을 얻기 위해서는 예제의 프로그램 코드에 대한 형식과 의미를 이해해야 하는데 예제의 제공 방식에 따라 그 이해도가 달라질 수 있기 때문이다. 이와 같은 현상은 특히, 프로그래밍 언어를 처음 학습하는 학습자들에게 더 두드러지게 나타날 수 있다. 따라서 프로그래밍 언어를 학습하는 학습자들로 하여금 예제의 프로그램 코드를 이해하는데 도움이 될 수 있는 최적의 방법이 모색되어야 한다.

2.2 유사성의 종류

예제를 통해서 프로그래밍을 한다는 것은 크게 두 단계로 나뉘어 진다. 첫째 단계는 예제의 프로그램 코드와 그 실행 결과간의 관계 즉, 어떤 프로그램의 코드 부분이 실행 결과의 어떤 부분과 관련되는가를 이해하는 단계이다. 프로그램 코드의 제공 방식은 바로 예제 프로그램 코드와 그 실행 결과간의 관계를 얼마나 쉽게 이해할 수 있게 하는가에 영향을 줄 수 있다.

둘째 단계는 예제 프로그램 코드와 그 실행 결과간의 관계를 이해한 것을 바탕으로 프로그래밍 과제의 실행 결과에 해당하는 프로그램 코드를 작성하는 단

1) 한편, 예제 중심의 프로그래밍은 예제에 의한 프로그래밍과는 구별될 필요가 있다. 예제에 의한 프로그래밍은 컴퓨터에게 컴퓨터가 처리할 자료의 사례들을 제공하여 시스템으로 하여금 자동적으로 사례 데이터로부터 프로그램을 유도해 내는 것을 의미한다. 반면에, 예제 중심 프로그래밍은 예제들을 시각적 보조 자료로 활용하거나 예제의 프로그램 코드의 전체 혹은 부분을 프로그래밍 과제에 대한 프로그램 코드에 활용하면서 프로그래밍하는 것으로 정의된다. 본 연구에서는 후자의 정의가 사용된다.

계이다. 이 단계에서는 예제의 실행 결과와 프로그래밍 과제의 실행 결과의 유사성이 문제가 될 수 있다. 그 둘 간에 어떤 종류의 유사성을 가지는가에 따라 프로그래밍 과제의 프로그램 코드를 작성하는 것이 쉬워지거나 어려워질 수 있을 것으로 예측된다

일반적으로 유사성은 여러 형태를 취할 수 있다. 이러한 형태들 중에서 가장 공통적인 것이 사물의 지각적(perceptual) 유사성을 의미하는 표면적 유사성과 내재적 속성의 유사성을 의미하는 심층적 유사성이다. 유사성에 대한 두 번째 접근은 전체적 유사성과 차원적 유사성이다. 이 차이는 두 사물을 전체적 관점에서 유사하게 보느냐 혹은 분별 가능한 특정 차원에 따라 유사하게 보느냐 하는 것이다. 따라서 표면적 유사성과 심층적 유사성은 차원적 유사성에 포함될 수 있다[11]. 그런 의미에서 보면, 예제의 실행결과와 프로그래밍 과제의 실행 결과 간의 유사성도 결국 전체적 비교보다는 프로그래밍 언어의 학습에 도움을 줄 수 있는 유사성의 차원을 찾아내 보려는 것이므로 차원적 유사성의 한 범주에 해당할 수 있다.

그렇다면 어떤 차원들이 프로그래밍 성과와 관련이 될 수 있는가를 알아볼 필요가 있다. 왜냐하면 만일 오해를 유발하는 표면적 유사성이 나타날 경우, 잘못된 유추가 적용되어 부정적 전이가 나타날 수도 있기 때문이다[4]. 따라서 프로그래밍 과제의 실행 결과와 예제의 실행 결과 간의 표면적 유사성이 유추적 프로그래밍에서 중요한 역할을 할 것으로 예측해 볼 수 있다. 한편, 프로그램이 어떻게 작동하는가를 이해하는데 초점을 맞춘 기능적 문제 해결은 학습의 향상과 관련된다는 기존 연구도 있다[9]. 이 두 가지 연구를 종합해 보면, 프로그래밍 언어 학습에 관련된 예제를 표면적 유사성과 프로그램의 작동을 의미하는 기능적 유사성 차원으로 나눌 수 있다. 여기서 프로그래밍 과제의 실행 결과와 예제의 실행 결과가 표면적으로 유사하다는 것은 외형적으로 보여지는 모습이 유사하다는 것을 의미하며 기능적으로 유사하다는 것은 사용자와의 상호작용에 따른 프로그램의 반응 방식이 유사하다는 것을 의미한다.

2.3 프로그램의 성과 측정

프로그래밍 과제의 결과인 작성된 프로그램 코드에 대한 평가는 구문적 점수와 의미적 점수로 이루어진다. 구문적 점수는 피험자들이 작성한 프로그램 코드를 컴파일하여 프로그램의 오류를 확인한 후에 그 오류들을 모두 디버깅하는데 필요한 최소의 단계로 정의된다. 예를 들어, 피험자가 프로그램을 작성할 때 }를 써 넣지 않아 컴파일러에서 여러 개의 오류가 발생한 것으로 나타난다 하더라도 }를 한 번만 삽입하면 그 오류들이 모두 제거될 수 있으므로 구문 점수는 1점만이 기록되도록 하였다. 이와 같은 방법으로 오류를 점수화하였으므로 점수와 프로그램 수행 성과는 서로 반비례의 관계를 가지게 된다.

의미적 점수는 일단 구문적 오류를 모두 제거한 상태에서 피험자가 작성한 프로그램 코드에 의한 프로그램 실행 결과와 주어진 프로그램 과제에서 보여지는 실행 결과와의 차이를 없애기 위한 최소의 단계로 정의하였다. 예를 들면, 피험자가 작성한 프로그램의 실행 결과에서 하나의 버튼 레이블만이 잘못된 경우에 의미적 점수가 1점이 된다. 만일 프로그램 코드에서 잘못 추가된 부분이 있는 경우에도 의미적 점수에 1점이 부가된다. 한편, 의미적 점수의 채점 단위는 추가 또는 삭제되어야 하는 프로그램의 줄 수로 계산되었다. 따라서 의미적 점수도 프로그램 수행 성과와 반비례의 관계를 가진다.

2.4 연구 모형

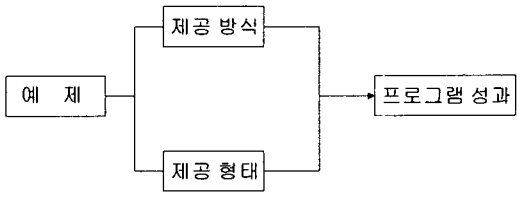


그림 1 연구 모형

본 연구의 연구 모형은 프로그래밍 학습에 예제가 사용될 때 예제를 제공하는 방식과 제공되는 예제의 형태가 프로그램의 성과에 영향을 미친다는 것이다. 예제의 제공 방식으로는 예제의 프로그램 코드에 관련된 설명을 추가적으로 제공하는 방식(「예제1+ 설명」)과 예제만을 두 개 제공하는 방식(「예제1+ 예제2」)을 사용하였다. 한편, 예제의 제공 형태에 있어서는 두 개의 예제를 제공할 경우(「예제1+ 예제2」)에 첫 번째 예제(예제1) 프로그램의 실행 결과는 프로그래밍 과제의 실행 결과와 외형적인 측면과 기능적인 측면에서 모두 유사한 예제를 제공하였고 두 번째 예제(예제2)는 외형과 기능에 따라 유사하고 상이한 네 가지 예제를 제공하였다.

2.5 연구의 가설

본 연구의 가설은 예제의 제공 방식과 제공 형태에

따른 프로그램 성과의 차이를 밝히는 것이다.

2.5.1 예제의 제공 방식

H1 예제를 두 가지 방식 즉, 「예제1+예제2」와 「예제1+설명」방식으로 제공 받은 집단 간에 프로그래밍 과제 수행에 차이가 없을 것이다.

H1a 「예제1+예제2」집단과 「예제1+설명」집단 간에는 프로그래밍 과제의 프로그램 코드에 대한 구문적 점수의 차이가 없을 것이다.

「예제1+ 예제2」집단은 두 예제의 프로그램 코드를 비교함으로써 프로그램의 형식에 대한 보다 확실한 파악이 가능하므로 프로그램 구문을 더 정확히 사용할 것이다.

H1b 「예제1+예제2」집단과 「예제1+설명」집단 간에는 프로그래밍 과제의 프로그램 코드에 대한 의미적 점수의 차이가 없을 것이다.

프로그래밍 과제에 맞게 프로그래밍을 하기 위해서는 예제의 프로그램 코드를 보다 정확히 이해하고 이를 응용하는 과정이 필요하다. 「예제1+설명」집단은 예제의 프로그램 코드를 단편적 설명에 의존해서 이해하려 할 것이기 때문에 프로그램의 전체적 의미를 파악하는데 어려움을 겪을 것으로 보이고 「예제1+예제2」집단은 프로그램 코드를 구성하고 있는 언어 요소들에 대한 응용 사례를 하나 더 제공받음으로써 과제 수행이 상대적으로 용이할 것이다.

H1c 「예제1+예제2」집단과 「예제1+설명」집단 간에는 프로그래밍 시간에 차이가 없을 것이다.

「예제1+설명」집단의 피험자들이 부가적 설명을 통해서 프로그램을 이해하려 할 경우, 대체로 주어진 모든 설명들을 읽으면서 이해하려는 경향을 보이기 때문에 내용을 읽고 이해하고 기억하는 과정에 많은 시간을 사용할 것으로 보인다. 반면에, 「예제1+예제2」집단의 경우 두 예제들만을 통해서 프로그램을 이해하기 때문에 예제 프로그램의 코드를 이해하는 범위가 프로그래밍 과제의 해결과 관련된 특정 부분으로 한정되어 보다 짧은 시간 안에 프로그램을 마칠 것으로 보인다.

2.5.2 예제 프로그램의 형태

H2 「예제1+예제2」집단들 간에 예제2의 실행 결과가 프로그래밍 과제의 실행 결과와 외형적으로 유사한 경우와 기능적으로 유사한 경우에 따라 프로그래밍 성과에 차이가 없을 것이다.

H2a 「예제1+예제2」집단들 간에는 예제2가 외형적이나 기능적으로 유사한 경우와 상이한 경우 에 따라 프로그래밍 과제에 대한 프로그램 코드의 구문적 점수의 차이가 없을 것이다.

피험자들은 주어진 두 예제 중에서 외형적으로 또는 기능적으로 상이하다고 판단되는 사례는 제외시킨 상태에서 프로그램을 작성할 것으로 보인다. 따라서 두 예제 모두 외형적 또는 기능적으로 유사한 예제로 판단된 경우에는 두 예제를 비교하면서 구문의 규칙을 파악할 수 있으므로 구문적 점수에서 더 좋은 성과를 보일 것이다.

H2b 「예제1+예제2」집단들 간에는 예제2가 프로그래밍 과제의 외형적 또는 기능적으로 유사한 경우와 상이한 경우에 따라 프로그래밍 과제에 대한 프로그램 코드의 의미 점수의 차이가 없을 것이다.

프로그래밍 과제에 대한 코드를 작성할 때 피험자가 실행 결과의 외형적인 부분에 대해서는 예제와 프로그램 코드를 비교적 쉽게 대응시킬 수 있겠지만 프로그램의 기능에 대해서는 예제 프로그램의 실행 결과와 프로그램 코드간의 대응이 다소 어려울 것이다. 그 이유는 프로그램의 기능을 파악하는 것이 프로그램과 피험자간의 상호작용을 통해서만 가능할 뿐 아니라 피험자의 입력의 종류에 따라 프로그램의 처리 방식이 달라져야 하므로 프로그램 코드가 다소 복잡해지기 때문이다. 이 경우에 기능적으로 유사한 두 예제가 제공되면 프로그램의 실행 결과와 프로그램 코드간의 관계 파악에 대해 더 많은 힌트를 제공할 수 있어 프로그래밍 과제를 수행하는 것이 쉬워질 것으로 예상된다.

H2c 「예제1+예제2」집단들 간에는 예제2가 외형적 또는 기능적으로 유사할 경우와 상이한 경우 에 따른 프로그래밍 시간의 차이는 없을 것이다.

예제2가 프로그래밍 과제와 상이하다고 판단될 경우 피험자는 예제2를 배제한 상태에서 예제1만으로 프로그래밍을 하게 될 것으로 예측된다. 이처럼 하나의 예제만을 활용하여 프로그래밍을 하게 되면 피험자가 이해해야 하는 프로그램 코드에 관한 정보의 양이 적어지는 반면에 프로그램 코드를 이해하는 데 필요한 힌트가 상대적으로 적어지게 된다. 그 반면에 두 예제를 모두 활용할 경우에는 이해해야 할 내용은 많아지지만 도움이 되는 힌트도 증가하게 되어 결국은 프로그래밍에 걸리는 시간은 감소할 것이다.

3. 실 험

본 실험은 예제의 제공 방식과 제공 형태에 따른 프로그래밍 성과의 차이를 알아보는데 그 목적이 있다. 실험은 두 집단으로 나누어 실시되었는데 첫 집단에서는 하나의 예제와 부가적 설명을 제공받은 통제 집단이 프로그래밍 과제를 수행하였다. 두 번째 집단에서는 외형과 기능에서 다른 예제를 하나 더 제공받은 실험집단이 프로그래밍 과제를 수행하였다.

프로그램 언어로는 객체지향 프로그래밍 언어인 JAVA를 사용하였다. JAVA 언어는 인터넷 환경에 잘 맞아 앞으로 인터넷을 통한 학습에 유리하다는 장점을 가지고 있다[4]. 게다가 프로그램의 재사용을 염두에 두고 만들어진 언어이므로 프로그래밍 효율이 높고 이미 만들어진 라이브러리 클래스들을 불러들여서 쓰게 되어 있기 때문에 프로그램 코드가 다른 언어에 비해 상대적으로 단순하여 프로그램을 이해하기에 용이하다는 장점도 있다. 특히, 프로그램 언어의 개발이 최근에 이루어져 아직 배우지 않은 사람이 많아 프로그램 초보자들을 구하기가 쉽다는 이점도 있다.

3.1 피험자

모두 46명의 피험자들이 실험에 참가하였으며 이들은 통제 집단과 실험 집단에 무작위로 할당되었다. 먼저, 11 명의 피험자들이 하나의 예제의 실행 결과와 프로그램 코드 그리고 각 프로그램 코드에 대한 부가적 설명을 제공받는 통제 집단에 할당되었다 (「예제1+설명」 집단). 그들은 실험에서 사용될 JAVA 프로그래밍 언어에 대한 사전 지식이 전혀 없었다. 통제 집단에 속한 피험자들이 작성한 프로그램을 평가한 후에 한 명의 피험자가 집단 평균에서 표준 편차의 두 배의 범위를 벗어나 분석에서 제외하였다. 이러한 조치는 지나치게 높거나 낮은 점수를 가진 소수의 피험자들이 전체의 결과를 왜곡시키는 것을 방지하기 위해서 취해졌다.

프로그램 코드에 대한 부가적 설명 대신에 예제가 하나 더 제공된 실험 집단은 JAVA 프로그래밍 언어에 경험이 없는 35명의 피험자들로 이루어졌다 (「예제1+예제2」 집단). 통제 집단과 마찬가지로 피험자들이 작성한 프로그램 코드를 평가하여 평균에서 표준편차의 두 배의 범위를 벗어난 여섯 명의 피험자들은 분석에서 제외하였다. 제외된 피험자들은 네 실험 집단에 고르게 분포되어 있었다. 표기 상의 명확성을 기하기 위하여 통제 집단은 「예제1+설명」 집단으로 실험 집단은 「예제1+ 예제2」 집단으로 명명하였다.

3.2 실험 과제

모든 실험 과제들은 피험자들이 필요할 때 보다 쉽게 참조할 수 있도록 컴퓨터 스크린을 통해 제공되었다. 통제 집단의 과제에 대한 인터페이스는 실험 집단의 인터페이스와 차이가 있었는데 이러한 인터페이스의 차이는 통제 집단에게는 예제의 프로그램 코드에 부가적 설명을 제공하였고 실험 집단에게는 별도의 설명 대신에 또 하나의 예제가 제공되었다는 점에서 비롯되었다. 즉, 통제 집단에게는 주어진 예제의 프로그램 코드마다 관련된 언어적 설명과 코드의 부분적 활용 사례를 제공하였다. 한편, 실험 집단에게는 두 예제의 프로그램 코드와 실행 결과를 한 화면에서 동시에 비교할 수 있도록 인터페이스를 제공하였다.

3.2.1 「예제1+설명」 집단

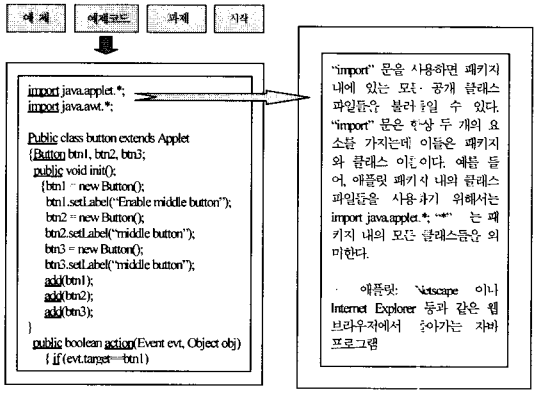


그림2. 예제 프로그램의 코드와 부가적 설명

그림 2는 「예제1+설명」 집단의 실험 과제를 보여주고 있다. 왼쪽 프레임 상단에 있는 버튼들은 다음과 같은 기능들을 가지고 있다. 예제 버튼은 예제 프

로그램의 실행 결과를 보여준다. 예제코드 버튼은 예제 프로그램의 코드를 보여주는 기능을 가지고 있다. 과제 버튼은 프로그래밍 과제의 실행 결과를 그리고 시작 버튼은 로그 시간을 기록하기 위해서 사용된다.

예제 프로그램의 코드에 포함된 언어적 구성요소를 설명하는데 11 개의 프레임이 사용되었다. 그림 1의 오른 쪽 프레임은 그 중 한 프레임을 보여주고 있다. JAVA 프로그램의 초보자들을 대상으로 한다는 점을 감안하여 설명은 가급적 쉽게 쓰여졌다. 각 프로그램 코드에 대한 설명과 부분적 활용 사례는 프로그램 코드의 중요한 요소들에 한해 제공되었는데 여기에는 'import', 'public', 'button', 'add', 'action' 그리고 'if' 등이 포함되었다.

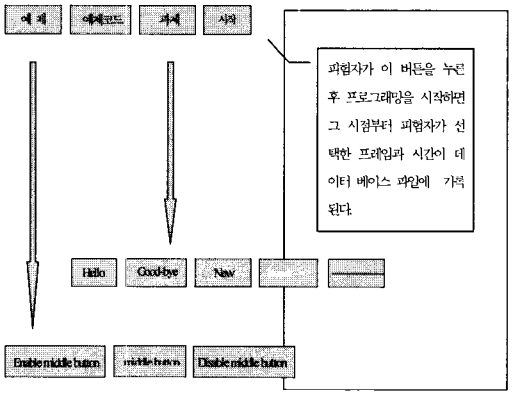


그림 3. 실험 자료에 포함된 각 통제 버튼들의 역할

그림 3에서는 예제에서 제공된 각 버튼의 역할을 보여주고 있다. 주어진 예제는 'Enable middle button', 'middle button', 그리고 'Disable middle button' 등의 세 가지 버튼을 가진다. Disable middle button은 가운데 버튼을 비활성화 시킨다. 버튼이 비활성화 되면 버튼 레이블의 색이 없어지고 음각화 되며 버튼을 눌러도 아무런 반응을 보이지 않게 된다. Enable middle button은 가운데 버튼을 다시 활성화시킨다. 버튼이 다시 활성화되면 버튼 레이블의 색이 다시 나타나게 되고 버튼을 클릭하면 눌려지는 반응을 보이게 된다.

프로그래밍 과제는 다섯 개의 버튼으로 이루어져 있다. 각 버튼은 버튼 레이블을 가지며 네 번째 버튼은 영구적으로 비활성화 되어 있다. 이 버튼은 그 오른쪽의 버튼에서 변환결과가 나타남을 가리키고 이 버튼이 포함된 이유는 버튼이 비활성화 되는 기능을 표현하기 위해서 이다. 앞의 세 버튼을 클릭하면 각 버튼에 해당하는 반응들이 다섯 번째 버튼에서 나타나게 된다.

3.2.2 「예제1+ 예제2」 집단

「예제1+ 예제2」 집단의 경우, 제공된 두 예제 모두 주어진 과제를 수행하는데 필요한 모든 프로그램 코드의 요소들을 포함하고 있다. 이 요소들은 버튼을 만들고 버튼에 레이블을 붙이고 버튼을 비활성화 시키는 작업과 버튼에 가해진 조작에 대해 반응을 일으키게 하는 작업 등과 관련되어 있다.

첫 번째 예제(예제1) 첫 번째 예제는 「예제1+ 설명」 집단에게 제공하였던 예제1과 동일한 예제였다. 이 예제는 「예제1+ 예제2」 집단에 속한 네 집단 모두에게 동일하게 제공되었다

두 번째 예제(예제2) 두 번째 예제는 네 가지 서로 다른 방식으로 제공되었다. 네 가지 방식은 주어진 프로그래밍 과제의 실행 결과와 예제2의 실행 결과간에 기능적 또는 외형적인 유사성에 기초하여 만들어졌다.

표 1은 「예제1+ 예제2」 집단에게 제공된 네 가지 종류의 예제들을 보여준다. 먼저, 외형과 기능 모두 유사한 예제는 버튼들로만 이루어져 있고 제일 왼쪽에 있는 버튼을 누르면 그 오른쪽에 있는 다른 두 버

기능 / 외형	유사	상이
유사	TV/VTR TV on VTR off	Create Delete Button
상이	1 mile 1 inch In meter	Male Female Reset

표 1. 두 번째 예제로 제공된 예제의 네 가지 형태

튼 위의 레이블이 변하게 된다. 즉, TV/VTR 버튼을 한 번 누르면 오른쪽 첫 번째 버튼의 레이블이 TV on으로 나타나면서 버튼이 활성화되고, 오른쪽 두 번째 버튼의 레이블이 VTR off로 나타나면서 버튼이 비활성화 된다. TV/VTR 버튼을 다시 한 번 누를 경우, 버튼의 레이블들은 각각 TV off와 VTR on으로 바뀌고 버튼이 서로 반대로 활성화된다.

두 번째로, 외형상으로는 유사하지만 기능상으로는 상이한 예제는 버튼들로만 구성되기는 하였지만 그 기능이 레이블을 변화시키는 것이 아니고 버튼을 생성하고 삭제하는 기능을 가지고 있다. 버튼생성 버튼을 클릭하면 오른쪽 끝의 버튼이 생겨나고 버튼삭제 버튼을 클릭하면 오른쪽 끝의 버튼이 삭제된다.

셋 째로, 외형은 다르게 보이지만 기능이 같은 예제는 버튼이 가진 기능을 체크박스로 구현하였다. 따라서 외형적으로는 프로그래밍 과제와 전혀 다르게 보이지만 기능은 동일하도록 예제를 만들었다. 즉, 하나의 체크 박스를 클릭하면 그 결과를 오른쪽에 있는 버튼에 보여주게 함으로써 결국은 프로그래밍 과제와 동일한 기능을 하도록 만들었다. 1 마일로 표시된 체크박스를 클릭하면 단위를 미터로 바꾼 결과를 오른쪽 버튼 위에 보여주게 되고 1 인치로 표시된 체크박스를 클릭하면 센티미터 단위로 표현된 수치가 오른쪽 버튼에 나타나게 된다.

끝으로, 외형적으로도 다르고 기능적으로 다른 예제에는 체크박스를 사용하였지만 그 기능은 체크박스를 클릭하는 것이 다른 버튼의 레이블을 변화시키는 것이 아니라 체크 박스는 선택만 할 수 있고 버튼을 이용해서 선택한 것을 수정할 수 있도록 하는 기능을 제공하였다. 즉, 체크 박스에 남, 여 중의 하나를 클릭한 다음 리셋 버튼을 누르면 앞서 선택한 것을 변경할 수 있게 하였다.

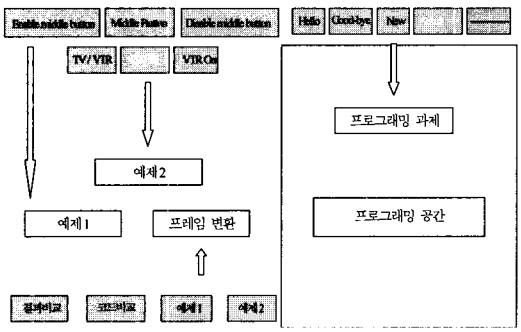
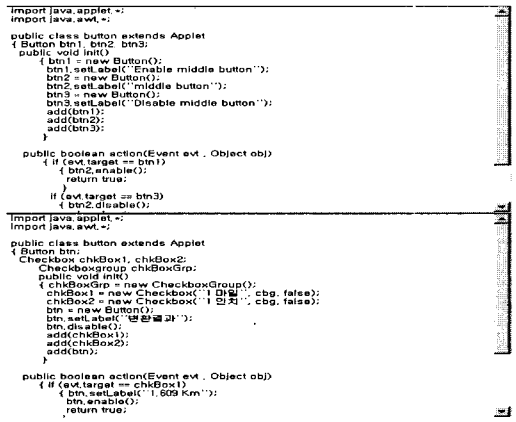


그림4. 두 예제 집단의 실험 자료

그림 4는 「예제1+ 예제2」 집단의 실험 자료를 보여준다. 실험자료는 컴퓨터 화면으로 제공되어 버튼이 눌려졌을 때의 반응을 직접 확인할 수 있도록 하였다. 오른쪽 프레임에는 과제가 출력 결과의 형태로 주어져 있다. 왼쪽 프레임의 상단에는 예제1과 예제2의 실행 결과를 동시에 보여주고 있다. 위쪽의 실행 결과가 예제1의 것이고 그 아래의 실행 결과가 예제2의 것이다. 왼쪽 하단에는 예제1의 실행 결과와 프로그램 코드, 예제2의 실행 결과와 프로그램 코드, 그리고 예제1과 예제2의 프로그램 코드 등을 볼 수 있게 해 주는 버튼들이 있다. 그림 3에서 제시되지 않은 세 가지 프레임은 뒤에서 자세히 설명할 것이다.

예제 프레임 「예제1+ 예제2」 집단은 프로그래밍을 하는데 사용될 수 있는 두 가지 예제를 네 가지 형태로 제공받는다. 이 네 가지 프레임들은 예제간의 비교나 예제 프로그램의 실행 결과와 프로그램 코드간의 관계를 쉽게 파악하도록 하기 위해서 제공된다. 네 가지 프레임은 예제1의 실행 결과와 프로그램 코드 프레임, 예제2의 실행 결과와 프로그램 코드 프레임, 예제1과 예제2의 프로그램 코드 프레임 그리고 예제1과 예제2 프로그램 실행 결과 프레임 등이다. 예제1과 예제2 프로그램 실행 결과 프레임은 그림 3에서 왼쪽 상단에 주어진 것과 같다. 피험자들은 이 프레임을 통해서 두 예제간의 실행 결과의 외형이나 기능들을 비교해 볼 수 있으며 또한 프로그래밍 과제의 실행 결과와도 쉽게 비교가 가능하다.



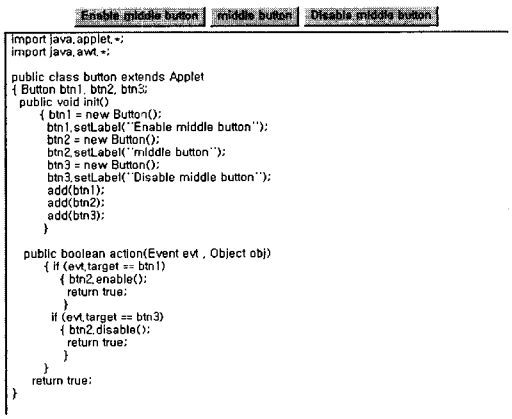
```
import java.applet.*;
import java.awt.*;

public class button extends Applet
{ Button btn1, btn2, btn3;
  public void init()
      { btn1 = new Button();
        btn1.setLabel("Enable middle button");
        btn2 = new Button();
        btn2.setLabel("middle button");
        btn3 = new Button();
        btn3.setLabel("Disable middle button");
        add(btn1);
        add(btn2);
        add(btn3);
      }

  public boolean action(Event evt , Object obj)
      { if (evt.target == btn1)
          { btn2.enable();
            return true;
          }
        if (evt.target == btn3)
          { btn2.disable();
```

```
import java.applet.*;
import java.awt.*;

public class button extends Applet
{ Button btn;
  Checkbox chkBox1, chkBox2;
    Checkboxgroup chkBoxGrp;
    public void init()
    { chkBoxGrp = new CheckboxGroup();
      chkBox1 = new Checkbox("1 마일", cbg, false);
      chkBox2 = new Checkbox("1 인치", cbg, false);
      btn = new Button();
      btn.setLabel("변환결과");
      btn.disable();
      add(chkBox1);
      add(chkBox2);
      add(btn);
    }

  public boolean action(Event evt , Object obj)
    { if (evt.target == chkBox1)
        { btn.setLabel("1.609 Km");
          btn.enable();
          return true;
```

그림 5. 프로그램 코드 비교 프레임

그림 5는 예제1과 예제2의 프로그램 코드를 동시에 볼 수 있는 프레임을 보여주고 있다. 이 프레임은 피험자들이 두 예제의 프로그램 코드를 동시에 비교할

수 있게 해 준다.




```
import java.applet.*;
import java.awt.*;

public class button extends Applet
{ Button btn1, btn2, btn3;
  public void init()
     { btn1 = new Button();
       btn1.setLabel("Enable middle button");
       btn2 = new Button();
       btn2.setLabel("middle button");
       btn3 = new Button();
       btn3.setLabel("Disable middle button");
       add(btn1);
       add(btn2);
       add(btn3);
     }

  public boolean action(Event evt , Object obj)
     { if (evt.target == btn1)
        { btn2.enable();
          return true;
        }
       if (evt.target == btn3)
        { btn2.disable();
          return true;
        }
     }
    return true;
}
```

그림 6. 예제의 출력 결과와 소스 코드

그림 6에서는 예제 프로그램의 실행 결과와 프로그램 코드를 함께 제공하는 프레임을 보여준다. 피험자들은 이 프레임을 통해서 예제 프로그램의 실행 결과와 프로그램 코드를 대응시키면서 프로그램 코드를 이해할 수 있다.

3.3 실험절차

1) 피험자의 인구통계적 데이터와 프로그래밍 경험에 대한 설문조사를 실시한다.

2) 피험자에게 실험목적과 실험과제에 대한 설명을 하고 실험 중 피험자가 수행해야할 verbal protocol에 관한 설명과 연습을 실시한다.

3) 피험자는 프로그래밍 과제를 자신이 원하는 순서와 방법에 따라 주어진 예제들을 사용하여 해결한다. 이 과정은 비디오 녹화 데이터와 system log 데이터로 기록된다.

4) 사후 설문서에서 프로그래밍 과정에서의 어려웠던 점등에 대해 질문을 하고 답변을 기록한다.

4. 실험 결과

실험 결과의 분석은 두 단계로 이루어졌다. 첫 단계에서는 피험자들의 프로그램 수행 결과를 분석하였고 두 번째 단계에서는 피험자들의 행위 데이터를 분석하여 프로그램 수행 결과에 나타난 차이의 원인을 밝혔다.

4.1 수행 자료 분석

프로그램 수행 자료의 분석은 먼저, 예제를 제공하는 방식에 따른 프로그램 수행 결과의 차이에 대한 분석을 하였고 다음으로 두 예제를 제공하는 집단들에 대해서만 예제의 형태에 따른 프로그램 수행 결과의 차이에 대한 분석이 이루어졌다.

4.1.1 예제를 제공하는 방식에 따른 가설의 검증

H1은 예제를 두 가지 방식 즉, 「예제1+예제2」와 「예제1+설명」방식으로 제공받은 집단 간에 프로그래밍 과제 수행에 차이가 있는가를 확인하기 위한 것이다.

H1a는 「예제1+예제2」집단과 「예제1+설명」집단 간에는 프로그래밍 과제의 프로그램 코드에 대한 구문적 점수의 차이가 없을 것이라는 귀무가설인데 t-test 분석의 결과, 「예제1+ 설명」집단과 「예제1+ 예제2」집단 간에는 구문적 점수에 있어서 유의한 차이를 보여 기각되었다(「예제1+ 설명」집단의 평균=7.61, 「예제1+ 예제2」집단의 평균 =2.85, t=3.01, p<0.01). 두 집단의 구문적 점수는 구문적 오류의 수를 나타낸 것이므로 낮을수록 수행결과가 우수하다고 볼 수 있다.

H1b는 「예제1+예제2」집단과 「예제1+설명」집단 간에는 프로그래밍 과제의 프로그램 코드에 대한 의미적 점수의 차이가 없을 것이라는 귀무가설로 t-test 결과 두 집단 간에 의미적 점수에 있어서는 유의한 차이가 없어 가설이 채택되었다.(「예제1+ 설명」집단 평균 =2.60, 「예제1+ 예제2」집단 평균 =2.14, t=0.40, p=n.s.). 두 집단의 의미적 점수는 의미적 오류를 나타낸 것으로 이 점수가 낮을수록 수행결과가 우수하다고 볼 수 있다.

H1a와 H1b에 대한 보다 상세한 분석을 위해서 「예제1+ 설명」집단이 각각의 「예제1+ 예제2」집단과 비교되었다.

두 예제 / 점수	외형	
	유사	상이
구문적	t=2.61 P=0.02*	t=3.33 P=0.01**
의미적	t=2.28 P=0.78	t=0.57 P=0.57

표 2. 외형적 측면에서의 점수 비교

두 예제 / 점수	외형	
	유사	상이
구문적	t=3.13 P=0.01*	t=2.76 P=0.02**
의미적	t=3.38 P=0.00**	t=0.91 P=0.37

표 3. 기능적 측면에서의 점수 비교

표 2와 표 3은 「예제1+ 설명」집단과 「예제1+ 예제2」집단들 간의 점수를 비교한 결과를 보여준다. 분석에 따르면 모든 「예제1+ 예제2」집단은 「예제1+ 설명」집단에 비해 프로그램의 구문적 오류를 덜 범한 것으로 나타났고 의미적 오류에 있어서는 예제2가 기능적으로 유사한 집단만이 「예제1+ 설명」 집단보다 우수한 수행을 보였다.

「예제1+설명」집단의 구문적 오류가 많이 나타난 것은 이 집단에 속한 피험자들이 프로그램 코드들을 비교해 볼 수 없기 때문에 예제1에서 주어진 구문 형식의 필요성을 확신하지 못하는 것으로 보인다. 반면에 「예제1+ 예제2」집단은 예제1과 예제2의 프로그램 코드를 동시에 비교하는 것이 가능하므로 프로그램 구문에 대한 더 많은 정보를 제공 받는 것으로 판단된다.

의미적 오류에 관해서 「예제1+ 예제2」집단 중 예제2가 기능적으로 프로그래밍 과제와 유사한 집단만이 「예제1+ 설명」집단에 비해 적은 수의 의미적 오류를 나타내었는데 이는 프로그램의 기능적 측면이 외형적 측면보다 프로그램 코드를 이해하는데 어려움이 있지만 프로그래밍 과제와 기능적으로 유사한 예제가 두 개 주어질 경우 피험자들은 프로그램 코드와 기능의 관계에 대해 더 많은 힌트를 얻을 수 있기 때문인 것으로 판단된다.

H1c는 「예제1+예제2」집단과 「예제1+설명」집단 간에는 프로그래밍 시간에 차이가 없을 것이라는 귀무가설인데 「예제1+설명」집단과 「예제1+예제2」집단 간의 전체 프로그래밍 시간의 차이를 분석한 결과 「예제1+ 예제2」집단이 프로그래밍에 훨씬 적은 시간을 사용한 것으로 나타나 기각되었다. (「예제1+ 설명」집단의 평균 = 58.58, 「예제1+ 예제2」집단의 평균 = 36.28, t = 5.10, p 〈 0.01).

요약하면, 「예제1+ 예제2」집단이 「예제1+ 설명」 집단에 비해 프로그램 구문 점수에서 우수한 결과를 보였으며 프로그래밍 시간도 훨씬 적게 사용한 것으로 나타났다. 한편, 프로그래밍의 의미적 점수에 있어서는 기능적으로 유사한 집단만이 「예제1+ 설명」 집단보다 뛰어난 성과를 보였다.

4.1.2 제공되는 예제의 형태에 따른 가설 검증

H2는 「예제1+예제2」집단들 간에는 예제2의 실행 결과가 프로그래밍 과제의 실행 결과와 외형적으로 유사한 경우와 기능적으로 유사한 경우에 따라 프로그래밍 성과에 차이가 있는가를 알아보는 것이다. 이를 위하여 「예제1+ 예제2」집단에서 예제2를 프로그래밍 과제와 기능적 측면과 외형적 측면에서의 유사성에 따라 네 종류로 나누어 네 집단에 제공하고 각 측면에 따른 프로그래밍 성과의 차이를 분석하였다.

H2a는「예제1+예제2」집단들 간에는 외형적이나 기능적으로 유사한 예제를 제공 받은 집단과 상이한 집단 사이에 프로그래밍 과제에 대한 프로그램 코드의 구문적 점수에 차이를 보이지 않을 것이라는 귀무가설인데 「예제1+ 예제2」 집단들 간에는 외형적 유사성 집단간 그리고 기능적 유사성 집단간에 구문적 점수에서 유의한 차이를 보이지 않아 가설이 채택되었다.

H2b는「예제1+예제2」집단들 간에는 예제2가 프로그래밍 과제의 기능적으로 유사한 경우와 기능적으로 상이한 경우에 따라 프로그래밍 과제에 대한 프로그램 코드의 의미 점수에 차이가 없을 것이라는 귀무가설인데 의미적 점수에 있어서는 기능적 유사 집단과 기능적 상이 집단간에 유의한 차이를 나타내어 가설이 기각되었다(유사 집단의 평균 =0.43, 상이 집단의 평균 =3.73, $F(1, 27) = 8.23$, $p<0.01$). 여기서 의미적 점수는 의미적 오류의 수를 나타내므로 예제2가 프로그래밍 과제와 기능적으로 유사한 집단이 상이한 집단보다 의미적 점수에서 더 높은 성취를 보인다.

H2c는「예제1+예제2」집단들 간에는 예제2가 외형적이나 기능적으로 유사한 경우와 상이한 경우에 따라 프로그래밍 시간의 차이가 없을 것이라는 귀무가설인데 프로그래밍의 전체시간에 있어서는 「예제1+ 예제2」집단들 간에는 유의한 차이를 보이지 않아 가설이 채택되었다. 이는 두 예제가 제공된 경우, 외형이나 기능의 유사성이나 상이성이 프로그램 시간에 영향을 미치지 않음을 보여준다.

요약하면, 「예제1+예제2」 집단 내에서는 예제2가 외형이나 기능의 두 측면에서 모두 주어진 프로그래밍 과제와 유사 또는 상이함에 따라 구문적 점수에 차이가 없는 것으로 나타났다. 그러나 예제2가 기능적 측면에서 프로그래밍 과제와 유사한 경우에는 상이한 경우보다 프로그래밍의 의미적 점수에서 뛰어난 성과를 보였다. 한편, 프로그래밍 시간에서는 집단들 간에 통계적으로 유의한 차이를 보이지 않았다.

4.2 행위 데이터 분석

수행 데이터 분석에서 나타났던 수행 결과의 차이를 설명하기 위하여 피험자들의 프로그래밍 과정에서 나타난 행위 데이터를 분석하였다.

4.2.1 「예제1+설명」 집단

표 4는 「예제1+설명」 집단에게 주어진 예제에 대한 설명 및 간단한 활용 사례의 프레임에 대해 각 피험자가 참조한 빈도의 평균을 나타내고 있다.

프레임	빈도(평균)
"Action"설명	2.45
"Action"활용 사례	3.18
"Add"설명	2.00
"Add"활용사례	2.00
"Button"설명	3.64
"Button"활용사례	3.09
"If"설명	2.45
"If"활용사례	2.64
"Import"설명	3.64
"Import"활용사례	2.73
"Public"설명	3.27

표 4. 한 예제 집단의 프레임 참조 빈도

이 분석의 목적은 수행 분석의 결과가 한 예제 집단에 속한 피험자들이 주어진 정보들을 충분히 활용하지 않은 데서 영향을 받았는지의 여부를 밝히는데 있다. 표 4의 결과는 모든 주어진 프레임들이 프로그램 도중에 최소 두 번 이상 참조되었음을 보여주고 있다. 이 결과로부터 「예제1+ 설명」 집단은 11개의 프레임을 통해 정보를 제공받았고 또 그 정보들을 참조했음에도 불구하고 단지 4개의 프레임을 통해 정보를 제공받은 「예제1+예제2」 집단보다 프로그램 성과가 떨어짐을 볼 수 있다. 따라서 예제를 제공할 때 정보의 양보다는 정보의 제공방식이 더 중요함을 알 수 있다.

4.2.2 「예제1+예제2」 집단

「예제1+예제2」 집단들간의 수행 결과에 대해 보다 자세한 원인을 밝히기 위해 참조 빈도와 참조 시간으로 표현되는 피험자들의 행위 데이터를 분석하였다.

프레임 참조 빈도율 각 프레임을 참조한 빈도를 보다 정확히 측정하기 위해서 개별 프레임의 참조 빈도를 전체 프레임 참조 빈도로 나눈 프레임 참조 빈도 비율을 사용하였다.

네 개의 프레임들은 크게 예제1과 예제2를 비교할 수 있는 예제간(inter-example) 비교 프레임과 예제1이나 예제2의 실행결과와 프로그램 코드의 관계를 볼 수 있는 예제내(intra-example) 비교 프레임으로 나누어 볼 수 있다. 후자는 피험자들이 예제 프로그램의 실행 결과와 프로그램 코드간의 비교를 용이하게 해 주고 전자는 피험자들이 예제1과 예제2의 실행결과간의 비교나 프로그램 코드간의 비교를 쉽게 할 수 있게 해 준다.

프레임 \ 측정치		빈도	
		외형	기능
예제간 비교	실행 결과	F(1.26)=2.53 (p=0.12)	F(1.26)=0.02 (p=0.88)
	코드	F(1.26)=1.95 (p=0.17)	F(1.26)=9.68 (p=0.00)
예제내 비교	예제1	F(1.26)=7.74 (p=0.01)	F(1.26)=9.66 (p=0.00)
	예제2	F(1.26)=0.00 (p=0.99)	F(1.26)=0.52 (p=0.48)

표 5. 프레임 참조 빈도의 비교

기능 집단 간에는 유사 집단이 상이 집단보다 두 예제의 프로그램 코드 프레임을 더 자주 참조하였다. (유사 예제 집단의 평균=0.42, 상이 예제 집단의 평균 =0.23, F=9.68, p 〈 0.01). 첫 번째 예제 프레임의 경우에는, 외형적 유사 집단 (유사 예제 집단의 평균 =0.18, 상이 예제 집단의 평균 =0.32, F=7.74, p 〈 0.01) 과 기능적 유사집단 (유사 예제 집단의 평균 =0.16, 상이 예제 집단의 평균 =0.32, F=9.66, p 〈 0.01) 모두 상이 집단 보다 상대적으로

적은 빈도로 참조하였다. 이 분석에서는 다른 유의한 결과는 발견되지 않았다. 또한 집단 간의 상호작용 효과도 없었다.

예제2의 외형적 측면과 기능적 측면의 유사성과 상이성에 따른 각 프레임들의 참조 빈도를 비교하기 위해서 각각의 측면에 따른 참조 빈도율을 비교하였다.

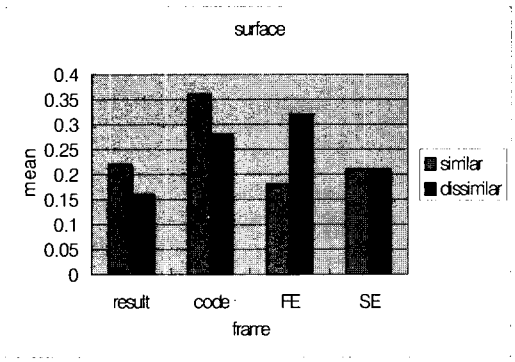


그림 7. 외형적 측면에서 참조 빈도율 비교

외형적 측면에서는 상이 집단이 유사 집단보다 첫 번째 예제(예제1) 프레임을 현저하게 많이 참조한 것으로 나타났다. 이는 예제2 프로그램의 실행 결과가 프로그래밍 과제의 실행 결과와 상이한 경우에 피험자들은 주로 외형적으로 프로그래밍 과제와 유사한 예제1을 참조하면서 프로그래밍을 한다는 것을 보여준다.

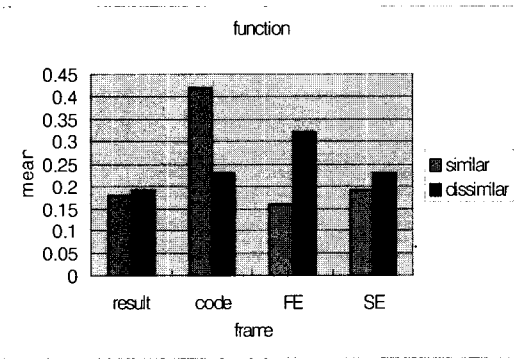


그림 8. 기능적 측면에서 참조 빈도율 비교

기능적 측면에서는 예제2가 기능적으로 유사한 집단의 경우 예제1과 예제2의 프로그램 코드를 비교하는 프레임을 더 많이 참조한 것으로 나타났는데 이러한 결과는 예제2가 기능적으로 상이할 경우 기능적으로 프로그래밍 과제와 유사한 예제1 프레임을 더 많이 참조한다는 것과 대조를 이룬다.

프레임 참조 시간 비율 각 프레임을 참조한 시간을 보다 정확히 비교하기 위해서 개별 프레임 참조 시간을 전체 프레임 참조 시간으로 나눈 시간 비율을 사용하였다.

프레임 \ 측정치		빈도	
		외형	기능
예제간 비교	실행 결과	F(1.26)=0.47 (p=0.50)	F(1.26)=0.20 (p=0.66)
	코드	F(1.26)=3.38 (p=0.08)	F(1.26)=3.44 (p=0.07)
예제내 비교	예제1	F(1.26)=5.03 (p=0.03)	F(1.26)=6.86 (p=0.01)
	예제2	F(1.26)=0.47 (p=0.50)	F(1.26)=0.78 (p=0.39)

표 6. 프레임 참조 시간 비교

외형적 유사 집단(유사 예제 집단의 평균 = 0.19, 상이 예제 집단의 평균 = 0.32, F= 5.03, $p<0.05$)과 기능적 유사집단 (유사 예제 집단의 평균 =0.17, 상이 예제 집단의 평균 =0.33, F=6.86, $p<0.05$) 모두 상이 집단에 비해 첫 번째 예제 프레임의 참조시간이 적었다. 다른 유의한 결과는 없었으며 상호작용도 나타나지 않았다.

예제2의 외형적 측면과 기능적 측면의 유사성과 상이성에 따른 각 프레임들의 참조 시간을 비교하기 위해서 각각의 측면에 따른 참조 시간의 비율을 비교하였다.

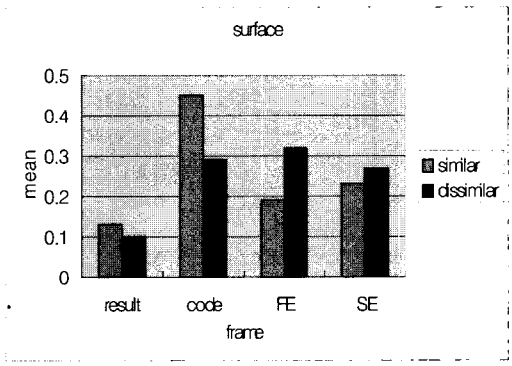


그림 9. 외형적 측면에서 참조 시간 비율 비교

프레임 참조 시간에 있어서 예제2가 외형적으로 프로그래밍 과제와 유사한 집단이 예제1과 예제2의 프로그램의 실행결과를 비교하는 프레임과 예제1과 예제2의 프로그램 코드를 비교하는 프레임을 더 오래 참조하였다. 그러나 예제1의 프로그램 실행 결과와 프로그램 코드를 보여주는 프레임과 예제2의 프로그램 실행 결과와 프로그램 코드를 보여주는 프레임은 상대적으로 짧게 참조한 것으로 나타났다.

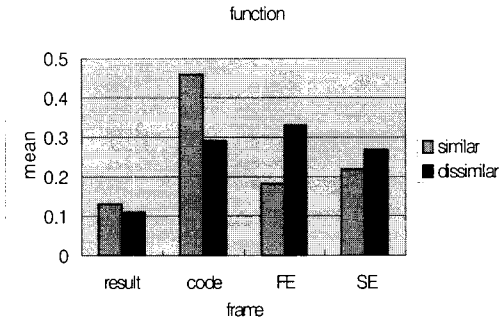


그림 10. 기능적 측면에서 참조 시간 비율 비교

예제2가 기능적으로 프로그래밍 과제와 유사한 집단에서도 그림 8의 외형적 유사집단에서 보인 것과 같은 형태를 나타냈다.

예제의 외형적 또는 기능적 유사성이 예제의 프로그램 코드를 더 많이 참조하게 된 이유는 다음과 같을 것으로 추정된다. 유사성은 앞서도 밝혔듯이 과제와의 관계에서 결정되므로 피험자가 판단할 때 두 번째로 주어진 예제가 외형적이든 기능적이든 과제와 유사하지 않다고 판단될 경우 덜 참조하게 될 것으로 보인다. 반면에 예제1의 경우 외형적이나 기능적으로 유사한 과제를 제공하였으므로 주로 예제1의 프로그램 실행 결과와 프로그램 코드를 많이 참조하게 된다고 볼 수 있다.

반면에 예제2가 외형적이든 기능적이든 주어진 과제와 유사하다고 판단된다면 과제 수행에 더 많은 실마리를 제공할 것으로 생각하게 되어 피험자들이 더 많이 참조하게 될 것이다. 하지만 프로그램 코드에 대한 아무런 설명도 제시하지 않았으므로 그 예제만 따로 참조하기 보다는 첫 번째 예제와 비교하면서 보는 것이 더 유리하다고 판단될 것이다. 두 예제의 프로그램 실행 결과를 비교하는 경우에는 외형에 대한 비교이므로 쉽게 차이를 구별할 수 있어 자주 참조할 필요성을 느끼지 못할 것이지만 프로그램 코드는 쉽게 이해하기 어렵기 때문에 더 많은 참조 횟수나 시간을 필요로 할 것이다.

그러나 이러한 설명들이 예제들의 프로그램 코드들을 서로 비교하는 행위가 프로그램의 의미 점수에 더 좋은 수행 결과를 나타내는 이유에 대한 직접적인 해석은 될 수 없다. 이에 대한 해석을 위해서는 먼저 예제를 활용해서 프로그램을 하기 위해서 거쳐야 되는 단계에 대한 지식이 필요하다. 예제를 활용한 프로그램은 기본적으로 세 단계를 거치게 된다. 먼저 피험자는 예제 프로그램의 코드의 어떤 부분이 그 실행 결과와 대응되는가를 구별해 내야 한다. 둘째로, 그 구별된 프로그램 코드를 프로그램의 실행 결과와 연관지어 이해하는 과정이 필요하다. 셋째로 두 번째 단계에서의 이해를 바탕으로 프로그래밍 과제에 대응되는 프로그램 코드를 작성하기 위해 예제 프로그램의 코드를 변형시키는 단계가 필요하다.

앞서 제시한 세 단계에 따라 외형적 측면이든 기능적 측면이든 예제가 프로그래밍 과제와 유사할 경우 더 많은 비교 행위를 유발시키지만 기능적으로 유사한 예제를 제공받은 집단만이 상이 집단에 비해 더 나은 수행을 보이게 되는 이유를 해석해 볼 수 있다. 예제의 실행 결과가 프로그래밍 과제의 실행 결과와 외형적으로 유사한 경우, 예제의 실행 결과에 나타난 외형적 형태와 대응되는 예제 프로그램의 코드 부분을 찾아내는 것이 비교적 용이하다. 따라서 하나의 예제만으로도 프로그램의 외형과 대응되는 프로그램 코드를 이해하는데 충분할 수 있다.

그러나 프로그램의 기능과 관련된 프로그램의 코드 부분은 사용자의 입력의 종류에 대응해서 프로그램의 출력을 제시해야 하므로 다소 프로그램 코드가 복잡하게 보일 수 있다. 즉, 프로그램 코드를 이해하기 위해서는 피험자가 프로그램에 대해 어떤 입력을 가했을 때 프로그램이 이에 대응하는 어떤 출력 결과를 보여주는가를 연결시켜서 이해하는 과정이 필요하다. 특히, 피험자의 입력에 대한 프로그램의 출력이 프로그램의 실행 결과에서는 동시적으로 발생하는 것처럼 보이지만 프로그램 코드는 이 과정을 순차적으로 처리하기 때문에 이 두 상황을 연결시키는 것이 어려울 수 있다.

만일 기능적으로 유사한 두 예제가 제공된다면, 먼저 비교를 통해서 대응하는 기능을 가능하게 하는 프로그램 코드 부분을 구별해 내기가 용이해 진다. 그 이유는 프로그램의 기능과 프로그램 코드간의 대응관

계를 이해할 수 있는 더 많은 힌트를 제공하기 때문이다. 또한, 이 경우에 피험자들은 기능의 유사한 변형이 어떻게 프로그램 코드에 반영되는가에 대한 더 많은 정보를 갖게 되므로 이를 이용하여 보다 쉽게 프로그래밍 과제를 수행할 수 있게 된다.

5. GOMS 모델

예제가 주어지는 두 가지 방식 즉, 「예제1+설명」과 「예제1+예제2」에 따라 프로그래머들이 프로그래밍을 해나가는 전체적인 인지과정을 모형화하게 되면 예제가 주어지는 방식에 따라 프로그래밍 과정이 어떻게 달라지는가를 보다 명확하게 이해할 수 있다. 따라서 본 연구에서는 피험자들의 행위 데이터를 기초로 인지 모형의 하나인 GOMS모형을 만들었다 [1]. GOMS 모형은 Goal, Operation, Method, Selection rules 들로 이루어진 모형으로 Goal은 피험자의 목표를 의미하고 Operation은 그 목표를 수행하기 위한 행동이다. 한편, Goal을 달성하는데 필요한 행동이 여러 가지 행위를 통해 가능하게 되는 경우에 목표 달성에 필요한 일련의 행위의 각 집합을 Method라하며 여러 Method 중에 하나를 선택하는 것을 Selection rule이라 한다[1].

5.1 「예제1+ 설명」 집단의 인지적 모형

통제 집단인 「예제1+ 설명」 집단의 행위를 GOMS 모델을 통하여 체계화시키면 아래와 같다.

GOAL: Write task program

· **GOAL: Understand**
· · **GOAL : Understand task problem**
· · · Read written task problem
· · · Find out functions through program output

· · **GOAL: Understand example problem**
· · · Find out functions through program output

· · **GOAL: Understand source program code**
· · · Read explanation of example constructs

· **GOAL: Program**
· · [select GOAL: Copy lines from program source code
· · · [select GOAL: compare between task and example
· · · · search for similarity
· · · · infer common part and copy
· · · GOAL: understand given example
· · · · read explanations of example
· · · · select program lines and copy]
· · **[select GOAL : Simple modification**
· · · [select GOAL: compare between task and example
· · · · search for differences
· · · · modify example source code
· · · · GOAL: understand example
· · · · read explanations and usage
· · · write task program source code]
· · **[select GOAL : Complex modification**
· · · [select GOAL: compare between task and example
· · · · map between example and source code
· · · · select appropriate command
· · · · modify command and place
· · · · GOAL: understand given example
· · · · read explanations and usage
· · · · select appropriate command
· · · · write task program source code]]
· **GOAL : Confirm program**
· · [select : simulate the program
· · read explanation and reflect]

과제 프로그램을 작성한다는 주목표는 세 개의 하위 목표를 가지게 된다. 먼저, 프로그램을 이해하고 (GOAL: Understand) 실제 프로그램을 작성하고 (GOAL: Program) 끝으로 작성된 프로그램을 확인 (GOAL : Confirm program)하는 것이다. 그 목표들은 구체적 행위에 의해 달성되게 되는데 프로그램을 하는 목표의 경우처럼 목표를 달성하는 방법이 여러 가지가 가능할 수 있다. 프로그램을 작성할 경우, 소스 코드에서 필요한 부분을 그대로 복사할 수도 있고 (GOAL: Copy lines from program source code) 조금 수정하여 사용할 수도 있고 (GOAL : Simple modification) 많은 수정을 가해야 할 경우 (GOAL : Complex modification)도 있다. 그 세 가지 방법들이 선택되는 규칙은 예제 프로그램의 소스 코드가 과제와 동일하다고 판단되면 그대로 사용하게 되고 동일 위치에서 변형이 가능하면 간단한 수정을 하여 사용하고 코드의 위치를 바꾸어야 하거나 예제와 다른 용도로 사용하는 경우에는 보다 복잡한 변형을 통해서 프로그래밍을 해 나가게 된다. 예를 들어 Import 문과 같은 것은 과제에서도 그대로 사용할 수 있어 그대로 사용하면 되는 예이고, 버튼의 레이블만 바꾸면 될 경우에는 단순 수정이 적용되는 예라고 볼 수 있다. 복잡한 변형을 가해야 하는 예에는

버튼의 레이블을 변환시켜야 할 때 이를 버튼의 레이블을 붙이는 방법을 활용하는 것이 해당된다.

5.2 「예제1+ 예제2」 집단의 인지적 모형

```
GOAL: Write task program
· GOAL : Understand
·· GOAL : Understand task problem
··· [select : look at appearance
····        find out functions]
··GOAL : Understand example problem
···[select look at appearance
···   find out functions]
··GOAL : Understand source program code
···[select compare given examples
···   refer to example 1]
· GOAL : Program
··[select GOAL : Copy program
···   Copy common lines of two examples
··  GOAL: Simple Modification.
···   [select GOAL Use two examples
····        refer to two-example frames
····        compare examples' appearance
····        modify example sources
···       GOAL Use example 1
····        refer to example 1 frame
····        examine example 1 appearance
····        modify example sources]
··  GOAL: Complex Modification.
···   [select GOAL Use two examples
····        refer to two-example frames
····        compare examples' functions
····        apply proper sources
···       GOAL Use example 1
····        refer to example 1 frame
····        examine example 1 functions
····        apply proper sources]
· GOAL : Confirm Program
··  Relate task source code to output
```

두 예제를 가지고 프로그래밍을 할 경우에도 프로그램의 순서는 한 예제와 동일하게 적용될 수 있다. 그러나 언어적 설명이 배제되어 있기 때문에 피험자들은 유추적 방법을 통해서만 프로그래밍을 할 수 있다. 예제의 유사성을 판단하는 방법은 크게 두 가지 존재한다. 하나는 프로그램의 외형적 유사성을 비교하는 방법이다. 주어진 예제가 과제와 공통적으로 버튼들로만 구성되어 있는 경우가 여기에 해당하며 이때 피험자는 주어진 예제가 유사하다고 판단한다. 또 다른 방법은 기능적인 유사성을 비교하는 것이다. 예제가 외형적으로는 달라 보일지라도 동일한 기능을 서로 다른 객체(프로그래밍 언어가 JAVA이므로)에 대하여 같은 행위 명령어를 사용하여 구현할 수 있다. 본 연구에서는 버튼의 레이블을 변환시키는 기능을 체크 박스를 이용해서 레이블을 변환하도록 만들 수 있다. 이 경우에 외형적으로는 달라 보이지만 결국 하는 일은 레이블을 변환시킨다는 동일한 기능을 수행하는 것이 된다.

따라서 예제 프로그램의 실행 결과는 프로그래밍 과제와 외형이나 기능에 대한 비교의 과정을 거쳐서 프로그래밍에 활용된다. 만일 예제2가 기능면이든 외형면이든 피험자의 관점에서 다르다고 판단되면 그 예제는 피험자가 프로그래밍을 하는 과정에서 배제될 수 있다. 그렇게 되면 피험자는 두 예제 중 기능적, 외형적으로 과제와 유사한 예제1을 통해서 프로그래밍을 하게 된다. 그러한 결정에 따라 피험자들은 두 예제를 비교하거나 예제1만을 참조하여 프로그래밍을 하게 된다.

5.3 두 집단의 인지적 모형 비교

「예제1+설명」과 「예제1+예제2」에 따른 두 가지 인지적 모형은 프로그래밍을 해나가는 목표의 구조(Goal Structure)는 동일한 형태를 띠고 있지만 목표를 달성하는 방법에 있어서는 차이를 보이고 있다. 즉, 「예제1+설명」 집단은 주어진 설명을 읽는 행위를 통해서 또는 예제1과 과제를 비교함으로써 주어진 과제에 해당하는 프로그램을 작성할 수 있고 「예제1+예제2」 집단의 경우에는 두 예제를 서로 비교하거나 외형과 기능 측면에서 주어진 과제와 유사한 예제1만을 통해서 프로그래밍을 할 수 있다.

앞서 제시한 두 가지 예제 제공 방식에 따른 프로그래밍 방법을 비교해 보면, 「예제1+설명」 집단과 「예제1+예제2」 집단 모두 예제 1만을 통해 프로그래밍을 할 수 있는 가능성이 있고 그렇게 프로그래밍을 했을 경우에는 프로그래밍 성과에 차이가 없어야 함에도 불구하고 프로그램 성과에 차이가 나는 것은 피험자들이 주어진 예제 제공 방식에 영향을 받는다는 것을 알 수 있다. 즉, 예제와 함께 설명이 주어지면 설명에 의존해서 프로그래밍을 하려하고 예제가

두 개가 주어지면 두 예제를 서로 비교하면서 프로그래밍을 하게 된다고 볼 수 있다.

6. 결론과 논의

본 연구의 결과는 세 가지로 요약된다. 첫째, 「예제1+ 예제2」 집단이 「예제1+ 설명」 집단에 비해 프로그램을 더 효율적으로 작성하였다. 의미적 오류는 차이가 없었지만 구문적 오류가 적었으며 더 짧은 시간 안에 프로그램을 작성하였다. 하나의 예제만 제공할 경우에는 피험자들이 주어진 설명의 이해를 바탕으로 프로그램을 하는 의미 중심적 프로그래밍이 이루어진다고 볼 수 있다. 반면에 두 예제만을 제공할 경우, 설명이 주어져 있지 않고 두 예제를 비교할 수 있도록 인터페이스를 제공하였기 때문에 유추적인 방식을 통해서 프로그램을 하게 됨으로써 위와 같은 차이가 발생했다고 사료된다. 둘째, 두 예제 집단들 간에는 외형적 유사성 측면에서는 유의한 프로그램 점수에 차이를 보이지 않았지만 기능적 유사성 차원에서는 의미적 점수가 상이 집단에 비해 유사 집단이 높게 나타났다. 이러한 결과로 볼 때, 두 개의 예제를 제공하는 것은 구문적 이해를 돕는 데는 효과적일지 모르지만 의미적 오류를 줄이기 위해서는 기능적으로 유사한 예제가 두 번째 예제로 제공되어야 함을 알 수 있다.

마지막으로, 「예제1+ 예제2」 집단들 간에는 예제2가 기능적으로 상이한 집단과 외형적으로 상이한 집단 모두 첫 번째 예제 프레임을 더 오래 참조한 것으로 나타났다. 또한 기능적 유사 집단과 외형적 유사 집단 모두 상이 집단에 비해 두 예제의 프로그램 코드를 더 빈번히 비교하였다. 반면에 상이 집단들은 첫 번째 예제 프레임을 더 자주 참조하였다. 이러한 결과는 외형적 유사성 뿐 아니라 기능적 유사성도 두 예제 간의 비교 행위를 유발시킨다는 것을 보여주고 있다. 그러나 프로그램의 외형을 구현하는 프로그램 코드는 비교적 단순한데 반하여 프로그램의 기능을 구현하는 프로그램 코드는 난이도가 상대적으로 높아 피험자들이 프로그램을 하는데 더 어려움을 겪게 되는데 기능적 유사 예제는 이에 대한 더 많은 단서를 제공하여 프로그램을 작성하는데 도움을 주는 것으로 판단된다. 두 번째 예제로 외형이나 기능적으로 상이한 예제를 제공받은 집단은 그 예제를 무시하고 첫 번째 예제로만 프로그램을 작성하는 경향을 보인다.

결론적으로, 두 예제를 제공하는 것이 한 예제와 부가적 설명을 제공하는 것보다 장점이 더 많은 것으로 나타났다. 이와 같은 결과는 두 예제를 동시에 비교하면서 프로그래밍을 할 경우 유추적인 방법을 더 쉽게 사용할 수 있다는 것을 간접적으로 시사한다. 즉, 유추적 방법에 따른 프로그래밍이 프로그램 코드를 언어적 설명을 통해 이해하면서 프로그래밍하는 것보다 더 효율적으로 프로그래밍을 하도록 유도하는 것으로 보인다. 예제 중심의 프로그래밍은 프로그래밍 과정에서 필요할 때마다 예제 프로그램을 참조할 수 있게 되므로 기억해야 될 사항을 줄여주어 초보 프로그래머의 인지적 부담을 상당히 경감시킨다. 하나의 예제와 부가적 설명을 제공하는 경우, 설명을 참조하지 않고도 유추적으로 프로그램을 작성할 수 있지만 대체로 설명에 의존해서 프로그램을 이해하려고 하였다. 따라서 많은 설명을 제공하려고 하는 것보다는 하나의 예제 특히, 기능적으로 유사한 예제를 제공하는 것이 유리하다.

본 연구는 여러 가지 연구의 제약을 가지고 있다. 첫째, 피험자들의 연령이 20대 초반으로만 구성되어 있다는 것이다. 연령이 편중되어 있어 일반화에 어려움이 있다. 둘째, 예제의 난이도를 조절하기 위해 노력은 하였지만 그러한 노력이 객관적으로 검증되지는 못 하였다. 셋째, 실험 방법이 개인 실험이었기 때문에 실험 시간이 많이 걸려서 피험자를 많이 확보할 수 없었다. 그러나 집단 실험 방법을 사용한다면 보다 많은 피험자를 확보할 수 있을 것이다.

참고 문헌

1. Card, S.K.., T.P Moran and Newell, A. The Psychology of Human-Computer Interaction, Lawrence Erlbaum Associates, Hillsdale: NJ, 1983.
2. 1. Gick, M.L. & Holyoak, K.J. (1983). Schema Induction and Analogical Transfer, *cognitive psychology 15*, 1-38.
3. Greenberg, S. and Witten, I.H.(1993). Supporting command reuse: mechanisms for reuse, Int. J. Man-Machine Studies 39: 391-425.
4. Linden, P.(1996). Just JAVA. The SunSoft Press JAVA series
5. Myers, B.A. Visual Programming, Programming by Example, and Program Visualization: A Taxonomy. In Porceedings of CHI' 86

Conference on Human Factors in Computing Systems (Boston, April 13-17). ACM, New York, 1986

6. Neal, L.R.(1989). A System for ExampleBased Programming. CHI '89 Proceedings.
7. Norman, D.A.(1993). Things that make us smart: defending human attributes in the age of the machine. Addison-Wesley.
8. Novick, L.R. (1988). Analogical Transfer, Problem Similarity, and Expertise. Journal of Experimental Psychology. Vol. 14,.No. 3, 510 520.
9. Pirolli, P. and M. Recker (1994). "Learning strategies and transfer in the domain of programming." 12: 235-275.
10. Spohrer, J.C. & Soloway, E. (1986). Analyzing the high frequency bugs in novice programs. In E. Soloway & S. Iyengar (Eds.), *Empirical Studies of Programmers* (pp. 230-251). Norwood, New Jerwey: Ablex Publinshing Corporation.
11. Vosniadou, S. & Ortony, A. (Eds.), (1989) Similarity and Analogical Reasoning. New York: Cambridge University Press.
12. Xinmin Zhu & Simon, H. A. (1988). Learning Mathematics from Examples and by Doing. In H.A. Simon(Ed.) Models of Thought vol II. New Haven, Ct.: Yale University Press.