Journal of Korea Multimedia Society Vol. 22, No. 4, April 2019(pp. 472-479)
https://doi.org/10.9717/kmms.2019.22.4.472

컴퓨터 비전공자 대상 SW 교육에서 컴퓨팅 사고력이 문제 해결 과정에 미치는 영향 분석

김 재 경†

Effect of Computational Thinking on Problem Solving Process in SW Education for non-CS Major Students

Jaekyung Kim†

ABSTRACT

Today, computational thinking takes an important role in problem solving in software education. As a result, software education as liberal arts for non-CS major students is rapidly expanding. It is necessary to study the effects of computational thinking on software problem solving ability compared to traditional programming language education. In this paper, we propose an evaluation model for analyzing the effects of computational thinking on the overall software development process, and analyze how the problem solving process is different for learners who take computing thinking classes and programming language courses as liberal arts courses. As a result, students who learned computational thinking showed higher ability in problem analysis and design process.

Key words: Computational Thinking, Programming Language, Problem Solving Process

1. 서 론

최근 가트너 그룹에 의하면 인공지능, 대화형 플랫폼, 그리고 몰입경험(Transparently immersive experiences) 등 소프트웨어 관련 기술이 미래사회를 이끌어나갈 핵심 전략으로 제시되고 있다. 국내에서는 소프트웨어 교육의 필요성에 따라 2016년도에 교육부와 미래창조과학부(현 과학기술정보통신부)에서 '소프트웨어 교육 활성화 기본 계획'을 심의하여 확정하였다. 개정 교육 과정에 따르면 소프트웨어 교육 인프라를 마련하고 초등학교는 '19년부터 17시간, 중학교는 '18년부터 단계적으로 34시간 이상 소프트웨어 교육을 필수화하고 있으며, 대학 교육에서도 소프트웨어 과목이 필수 과목으로 확대되는 추세이다.

현재 소프트웨어 교육은 컴퓨터과학 분야의 문제해결 기술인 컴퓨팅사고에 따라 문제를 논리적이고 효율적으로 해결할 수 있는 능력을 교육하는 방향으로 자리를 잡고 있으며, 교육용 프로그래밍 언어, 로봇 제어 및 피지컬 컴퓨팅, 웹 기반[1] 교육 및 앱 제작[2] 등의 다양한 방식으로 컴퓨팅사고를 교육하고 있다[3].

컴퓨팅사고는 추상화, 문제 분할, 데이터 수집 및 표현 및 알고리즘 등의 개념을 이용하여 실세계의 문제를 자동화된 컴퓨팅 기기로 해결할 수 있는 형태로 표현하는 사고 과정이므로, 컴퓨팅 사고력 교육의 핵심은 학습자의 문제 해결 역량을 강화하는 데 있다. 컴퓨팅 사고력을 기르기 위해서는 컴퓨팅사고 이론을 암기하는 데 그치거나 혹은 단순히 프로그래밍 언어의 문법을 학습하는 코딩 위주의 교육을 하는

※ Corresponding Author : Jaekyung Kim, Address: (21983) Songdokwahak-ro 85, Incheon, Korea, TEL : +82-32-749-3164, FAX : +82-32-749-3164, E-mail : kim.jk@yonsei.ac.kr

Receipt date : Jan. 10, 2019, Revision date : Mar. 15, 2019
Approval date : Mar. 26, 2019
† University College, Yonsei University

것이 아니라, 문제의 해결 과정을 프로그래밍 언어와 같은 도구를 이용하여 자료를 분석하고 적합한 알고리즘을 설계 및 구현하고, 컴퓨팅 기기를 이용하여 자동화된 문제의 해를 산출 수 있도록 하는 것이 필요하다.

특히 컴퓨팅 개념과 기술이 인문학, 사회과학, 의학, 경영학 및 이학 등의 모든 분야에 융합되고 있는 현 산업에서 컴퓨팅사고를 통한 프로그래밍 교육은 컴퓨터과학 비전공자들이 컴퓨팅 문제 해결 역량을 키우는 효과적인 방법이다[4].

그러나 컴퓨팅사고의 개념들을 장기간에 걸쳐 학습하고 다양한 분야의 문제 해결을 프로그래밍을 통하여 경험하는 컴퓨터과학 전공생들과 달리, 비전공생의 경우 한정된 시간에 컴퓨팅사고 개념과 프로그래밍을 학습하여 해당 전공 분야의 문제 해결에 적용하여야 하는 어려움이 있다. 따라서 비전공생을 대상으로 하는 컴퓨팅사고 교과목에서는 컴퓨팅사고 이론과 실제를 효율적으로 구성하여 교육하여야 한다.

최근 대학 교육에서의 비전공자를 대상으로 한 컴퓨팅사고 교육 과정이 활발히 개설되고 교육 효과에 대한 연구가 이루어지고 있으나[5, 6], 전통적인 프로그래밍 교육과 비교하여 컴퓨팅 문제 해결력에 관한 연구는 미흡한 실정이다. 본 논문에서는 프로그래밍 언어를 통한 문제 해결 과정에서 컴퓨팅사고 개념이 어떤 영향과 효과를 주는지 평가하여 일반적인 프로그래밍 언어 학습과의 차이를 알아보도록 한다. 이를 위해 컴퓨팅사고 개념을 학습하고 이를 문제 해결 과정에 적용하고 그 결과를 프로그래밍 언어로 표현하는 교과목을 수강한 학습자와, 전통적인 프로그래밍 언어 교과목을 학습한 학습자의 문제 해결 과정을 분석하여 차이점을 평가 및 비교하도록 한다.

본 논문의 구성은 다음과 같다. 2장에서 컴퓨팅사고 교육 효과를 평가하는 선행 연구에 대해서 살펴보고, 3장에서 학습자의 문제 해결 능력을 분석하기 위한 평가 모델을 제안한다. 그리고 4장에서는 제안한 모델에 따라 실험 그룹과 통제 그룹을 평가하여 분석 결과를 서술하고, 5장에서 결론을 맺는다.

2. 관련 연구

2.1 컴퓨팅 사고력 평가

컴퓨팅 사고력 교육의 효과를 측정하기 위해서 프로그래밍 언어나 소프트웨어 도구의 활용 없이 개념 습득과 사고 과정에 초점을 두고 학습자의 역량을 평가한 연구는 아래와 같다.

Rodriguez 등[7]은 검색 알고리즘, 이진수, 최소신장트리, 유한 상태 오토마타 개념을 학습하기 위한 언플러그드 활동을 개발하고, 학습자가 개발된 활동을 통하여 추상화, 데이터 표현, 그리고 알고리즘에 대한 개념 습득을 설문으로 측정하였다.

Gouws 등[8]은 컴퓨팅 사고력을 모델과 추상화(M&A), 패턴과 알고리즘(P&A), 도구와 자원(T&R), 처리와 변환(P&T), 추론과 로직(I&L), 평가와 개선(E&I)의 6가지 분류로 정의하고, 컴퓨터 올림피아드(Computer Olympiad)의 '펜과 종이(Pen-And-Paper)' 형식의 문제들을 훈련하도록 하여 대학 1학년 학생들의 컴퓨팅 사고력을 분석하였다.

Bort & Brylow[9]는 국내에서도 자주 사용되고 있는 CSTA에서 정의한 9가지 컴퓨팅사고 개념에, Connection to Other Fields 개념을 추가하여 총 10개의 기본 개념에 대한 평가 루브릭(Rubric)을 정의하였다. 학습자에게 컴퓨팅 사고력 개념 개념을 교육시킨 후 발표를 시키고, 이를 비디오로 녹화하여 각 개념에 대한 점수를 산정하였다.

국내 연구의 경우는 심순화 등[10]이 CSTA의 분류에 기초하여 9개 부문으로 이루어진 서답형 문항으로 컴퓨팅 사고력 검사를 실시하였으며, 권정인 및 김재현[11]은 컴퓨팅사고 과목을 수강한 대학교 신입생을 대상으로 CSTA의 분류에 기초한 총 51문항을 설문하여 컴퓨팅 사고력에 대한 결과를 평가하였다.

2.2 프로그래밍 구현물을 통한 컴퓨팅 사고력 평가

컴퓨팅 사고력으로 해결된 문제의 산출물인 소프트웨어 구현물을 분석하여 학습자의 컴퓨팅 사고력 향상을 평가하는 연구는 활발히 진행되고 있으며 국내외에 다음과 같은 사례가 있다.

Romero 등[12]은 비CS전공 학부생을 대상으로한 SW 과목에서 학습자들이 작성한 스크래치 산출물을 Dr. Scratch 도구를 이용하여 평가하였다. Dr. Scratch는 스크래치 결과물을 자동으로 분석하여 7개 항목(abstraction, parallelism, logic, synchronization, flow control, user interactivity, and data representation)에 대하여 수치를 산출한다.

김수환[13]은 컴퓨팅 사고력 교육에서의 학습자

평가를 위한 방안으로 학습자가 구현한 스크래치 코드를 분석하는 방법을 제안하였다. 동작, 제어, 형태, 관찰, 소리, 연산자 및 변수 블록이 구현물에서 사용된 빈도를 측정하여, 논리적사고, 플로우제어, 데이터표현, 병렬, 추상화, 동기화, 병렬과 같은 요소들에 대한 교육 효과를 학습자 수준에 따라 분석하고 평가하였다.

최형신[14]은 컴퓨팅 사고력을 6개(절차 및 알고리즘, 병행화 및 동기화, 자료 표현, 추상화, 문제 분해, 시뮬레이션)로 분류하고 스크래치로 구현한 산출물에서 변수와 리스트 사용, 함수 블록의 패러미터 사용 여부 등을 분석하여 학습자를 기초, 발달 및 능숙 단계로 평가하였다.

국내·외 연구는 초·중등 교육에 관한 연구가 활발히 진행되고 있어 구현 도구로서 스크래치를 이용하는 경우가 많았다. 특히 해외 연구의 경우 자동화된 평가 소프트웨어를 이용하여 스크래치 결과물을 분석하는 사례가 많았으며, 국내 연구의 경우는 연구자가 정의한 루브릭에 따라 블록을 분석하여 평가하는 사례가 많았다.

그러나 대학교 교육과정에서는 학습자가 자신의 전공 분야의 문제를 컴퓨팅 사고력으로 해결할 수 있도록 훈련하는 것이 요구되기 때문에 범용 프로그래밍 언어 및 도구를 활용하여 문제를 해결하는 것이 필요하다. 특히 컴퓨팅 사고력에 대한 지식 혹은 프로그래밍 산출물에만 국한하여 학습자의 능력을 평가하기보다는 문제의 분석 단계부터 디자인 및 구현 과정을 모두 포함하여 평가하고 교육 효과를 비교하는 것이 필요하다. 또한, 기존의 프로그래밍 언어 과목과 비교하여 컴퓨팅 사고력이 문제 해결에 어떤 영향을 미치는지 분석하는 것이 요구된다.

3. 제안 평가 모델

3.1 컴퓨팅사고 평가를 위한 모델 정의

소프트웨어 개발에서 문제를 해결하는 전 과정은 소프트웨어 개발 생명주기(SDLC: Software Development Life Cycle)로 구성되어 있다. SDLC는 개발 환경이나 규모에 따라 여러 단계로 세분화되나 일반적으로 문제 분석과 설계, 구현, 테스트 및 운영을 기본 단계로 사용한다[15].

본 연구에서는 문제 해결 단계별로 학습자를 평가하기 위해 소프트웨어 개발 단계를 기준으로 컴퓨팅 사고 개념을 분류하여 평가하도록 한다. 단, 학습자의 구현물을 실제로 배포하는 것은 아니므로 운영 단계를 제외한 Table 1과 같이 문제 분석과 설계, 프로그램 구현 및 테스트 단계를 사용하도록 한다.

각 단계에서의 컴퓨팅 사고 평가 항목은 CSTA의 정의를 사용하였으며 평가 문제에서 쓰레딩(Threading)과 같은 병렬 수행(Parallelization) 기능은 사용하지 않았으므로 제외하였다. 제안 평가 모델은 Table 2와 같으며 각 항목의 학습 목표를 얼마나 만족하느냐에 따라 전문가가 0점부터 3점까지 등급을 평가하도록 하였다.

분석 단계(Analysis)의 자료 수집(Data collection) 요소에서 평가자는 상태표(State table)를 이용하여 문제에 필요한 기능 및 자료들을 완전한 형태로 작성하여 이를 기준으로 학습자의 작성 결과를 평가한다. 자료 분석(Data analysis)에서는 문제 해결에 필요한 패턴과 자료의 연산이 정의되었는가를 판단한다. 자료 표현(Data representation)에서는 변수에 의미적(semantic)인 이름을 부여하고 변수의 유형과 자료 구조를 결정한다.

설계 단계(Design)에서는 문제 해결에 집중 할 수 있도록 추상화된 기능적 요구사항(Functional requirements)을 정의하고, 문제를 하향식 설계로 분할하여 도표로 표현하도록 한다.

구현 단계(Implementation)에서는 설계 과정에서 도출된 결과가 프로그래밍 명령문으로 잘 작성되었는가를 분석한다. 프로그래밍 산출물은 올바른 변수

Table 1. Steps for Evaluating Problem Solving Process

Step	Description
Analysis	Analyses user needs and define a clear functional requirements document
Design	model computational and organizational processes to show the overall flow of control
Implementation	Implementation of the software, corresponding to the design process
Test	Test if requirements as specified in the functional requirements

Table 2. Proposed Evaluation Model

Steps	CT Concepts	Results	Evaluation Elements
Analysis	Data Collection	Document	Gathering information appropriately
	Data Analysis	Document	Finding patterns and equations
	Data Representation	Document	Making sense of data and structure
Design	Abstraction	Document	Reducing complexity to focus on essential element of problem
	Decomposition	Document	Breaking problem into sub tasks
Implementation	Algorithm	Program	Writing correct steps for solving problem
	Automation	Program	Completing programs without run-time errors and exceptions
Testing	Simulation	Data	Finding a correct solution using program

의 사용, 올바른 제어 흐름과 문제 분할 개념에 따라 사용자 정의 함수를 이용하여 모듈화를 이루어야 한다. 또한 자동화(Automation)에서는 완성된 프로그램이 컴퓨팅 기기에서 오류 없이 실행되어야 한다.

마지막으로 테스트 단계에서 구현물이 요구사항을 모두 만족하고 올바른 해를 도출 여부를 판단하기 위해 정의된 테스트 케이스를 입력하여 평가한다.

4. 실험 결과

4.1 평가 및 분석

컴퓨팅 사고력이 프로그래밍을 통한 실제 문제 해결 과정에 어떠한 영향을 미치는지 알아보기 위해 대학교의 16주 정규 수업인 컴퓨팅사고 과목을 수강한 학습자를 실험집단(E.G.)으로, 동일한 학기에 프로그래밍 언어 과목을 수강한 학습자를 통제집단(C.G.)으로 구성하였다. 각 집단은 컴퓨터과학 비전공생 1학년 35명이며 기존에 컴퓨터 관련 수업을 수강한 적이 없었다.

컴퓨팅사고 과목에서는 컴퓨팅사고 개념을 학습하고 주어진 문제의 해결을 Python 언어로 구현하도록 수업을 구성하였으며, 프로그래밍 언어 과목에서는 Python 언어의 문법과 명령어의 활용으로 프로그램을 작성하는 데 초점을 두고 교육을 진행하였다.

실험집단과 통제집단의 사전 지식 검사는 문제 해결 과정에 대한 개념과 프로그래밍 경험을 묻는 내용으로 구성하여 측정하였다. 그 결과 Table 3와 같이 실험 그룹과 통제 그룹은 유의미한 결과를 보이지 않았다.

계획된 수업 과정을 모두 학습하고 각 집단은 주어진 동일한 실험 문제에 대한 해결 절차를 2시간 이내에 작성하도록 하였으며, 한 학기 동안 학습한 내용을 모두 포함하도록 사용자 입력 및 출력, 복수의 자료 유형과 리스트 구조, 다중 분기 및 반복 제어, 모듈화, 문자열 및 파일 처리를 사용하도록 실험 문제를 구성하였다.

피실험자는 문제 분석과 설계 과정을 문서로 작성하고, Python으로 프로그램을 구현하여 제출하도록 하였다. 제출물은 제안 모델에 정의된 항목을 전문가 3인이 평가하였으며 결과는 Table 4와 같다.

분석 단계에서 실험집단은 문제를 분석하고 필요한 자료와 자료 유형을 대부분 기술하였으나, 통제집단은 요구되는 자료의 일부만 작성하거나 문제 분석의 의미를 잘 이해하지 못하는 예도 있었다. 자료를 표현하는 방식에서도 실험집단은 추상적인 자료 표현 개념에 근거하여 자료에 의미적인 이름을 부여하는 경향이 있었으나 통제집단은 a, b, c와 같은 무의미한 자료 이름을 사용하거나 매우 짧은 길이의 단어를 사용하여 자료의 의미를 파악하기가 어려운 경우가 많았다. 그 결과, 분석 단계의 각 평가 항목들에서

Table 3. Pre-test Result

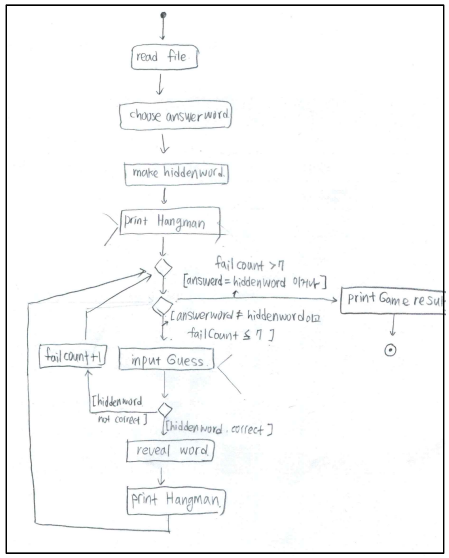
	N	Avg.	SD	t	df	p
E.G.	35	0.984	0.397	0.315	68	0.754
C.G.	35	0.950	0.507			

Table 4. Evaluation Results of Experimental and Control Groups

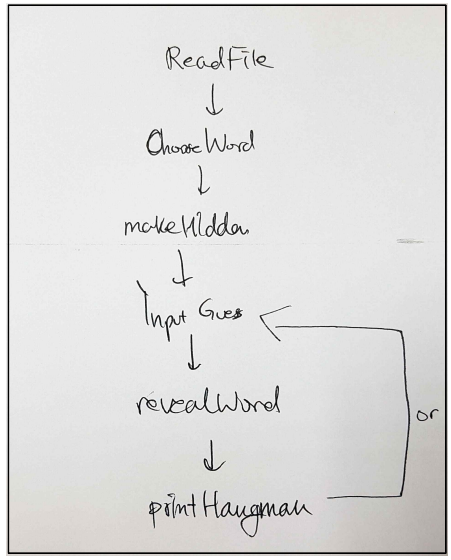
Steps	Elements	Groups	Avg.	Std.	t	p
Analysis	Data Collection	E.G	2.39	0.70	2.373	0.023
		C.G	1.83	0.71		
	Data Analysis	E.G	2.61	0.61	3.806	0.001
		C.G	1.83	0.62		
	Data Representation	E.G	2.39	0.70	3.382	0.002
		C.G	1.72	0.46		
Design	Abstraction	E.G	2.39	0.61	2.528	0.016
		C.G	1.83	0.71		
	Decomposition	E.G	2.17	0.70	2.243	0.032
		C.G	1.61	0.78		
Implementation	Algorithm	E.G	2.56	0.51	-1.027	0.312
		C.G	2.72	0.46		
	Automation	E.G	2.72	0.46	0.692	0.494
		C.G	2.61	0.50		
Testing	Simulation	E.G	2.39	0.70	-0.253	0.802
		C.G	2.44	0.62		

실험집단이 높은 점수를 얻었으며 두 집단은 유의미한 차이를 보였다.

설계 단계에서 두 집단은 기능적 요구사항을 추상적으로 정리하였다. 결과물을 평가한 결과 실험집단이 통제집단보다 요구사항을 대체로 명확하고 완전하게 작성하였다. 문제 분할의 경우 Fig 1(a)와 같이 실험집단은 문제를 기능 단위로 분할하여 전체적인 문제 해결의 흐름을 작성하였다. 각 행동은 추상적 형태로 정의되어 전체 프로그램의 실행 구조를 이해하기가 용이하였다. 이에 반해 통제집단은 Fig. 1(b)



(a) Experimental Group



(b) Controlled Group

Fig. 1. Decomposition Diagram of Design Step.

와 같이 문제 분할이 다소 간단하게 이루어지는 경향이 있어 전체 실행 구조를 이해하기가 어려웠다. 경우가 많았다. 설계 과정의 문제 분할은 Fig. 2의 점선으로 채워진 상자와 같이 사용자 함수의 정의로 구현이 되었으며 실험집단은 설계 과정의 문제 분할 결과에 따라 함수를 정의하여 Fig. 1(a)와 같이 모듈화된 프로그램을 작성하였다. 이에 비교해 통제집단은 Fig. 2(b)와 같이 모듈화가 비교적 낮은 프로그램을 구현하였다. 다만, 두 집단 모두 분할 사고 항목의 평균은 다른 항목에 비교해 다소 낮은데 문제 해결의 경험이 적은 입문자에게는 어려운 사고 과정으로 파악된다.

구현 과정에서 두 그룹이 작성한 알고리즘은 올바른 논리로 작성되어 평균에 유의미한 차이가 없었다. 또한, 구현물의 검증 과정에서 테스트세트를 이용하여 평가한 결과, 두 집단이 모두 양호한 실행 결과를 보였으며 프로그램의 입력 및 출력 결과의 평가에서 큰 차이가 없었다. 구현과 테스트 단계에서 차이가 발견되지 않은 것은, 실험 문제가 주어진 시간 내에 충분히 해결 할 수 있는 난이도로 설계되어 프로그램을 완성하는 데 어려움이 없었던 것으로 관찰되었다.

5. 결 론

소프트웨어 교육의 활성화와 함께 컴퓨팅사고 및 프로그래밍 관련 교과가 최근 몇 년 사이에 빠르게

```python
def selectWord():
    f = open("C:\\work\\fruits.txt")
    wordlist = []
    for line in f:
        wordlist.append(line[:-1].lower())
    f.close
    word = random.choice(wordlist)
    return word

def makeList(word):
    lst = []
    for letter in word:
        if letter not in lst:
            lst.append(letter)
    return lst

def maskWord(word, playeranswerlist):
    nomask = playeranswerlist
    for letter in word:
        if letter not in nomask:
            word = word.replace(letter, "_")
    return word

def printWord_History(playeranswerlist, historylist, targetword):
    print(maskWord(targetword, playeranswerlist))
    if historylist == []:
        print("")
    else:
        for x in historylist:
            print(x, end="")
        print("")

def printHangman(errorcount):
    printlist = ["|", "O", "/", "|", "\\", "/", "\\"]
    for i in range(6, errorcount - 1, -1):
        printlist[i] = " "
    print("""
          +---+
          {}  |
          {}  |
         {}{}{}  |
         {} {}  |
              |
         -----------
""".format(printlist[0], printlist[1], printlist[2], printlist[3],
printlist[4], printlist[5], printlist[6]))

def Success():
    print("Congratulation\nYou Saved the hangman!")

def Gameover(targetword):
    print("Hangman died... T_T\nThe answer is
{}".format(targetword))
```

(a) Experimental Group

```python
history = []

with open("fruits.txt", "r") as fruits_file:
    fruits_list = fruits_file.read().splitlines()

from random import randint

fruit_index = randint(0, len(fruits_list) - 1)
word = fruits_list[fruit_index].lower()
first_char_index = randint(0, len(word) - 1)

print(word)

entered_value = ""
first_char = word[first_char_index]

for i in range(len(word)):
    if word[i] == first_char:
        entered_value += first_char
    else:
        entered_value += "_"

def show_entered_value():
    print(entered_value)

def show_history():
    print("".join(history))

def update_entered_value(c):
    new_str = ""
    for i in range(len(entered_value)):
        if entered_value[i] != "_":
            new_str += entered_value[i]
        elif word[i] == c:
            new_str += c
        else:
            new_str += "_"
    return new_str

count = 0  # lose if count>=7
hangman_num = 0

while count < 7:
    show_entered_value()
    show_history()
    guess_char = input("Enter a guess: ")
    if not guess_char.isalpha():
        print("Incorrect input.")
        continue
    elif guess_char in history:
        print("Incorrect input.")
        continue
```

(b) Controlled Group

Fig. 2. Modularization Level of Program.

늘어나고 있으며, 구체적인 수업 구성과 효과에 대해서는 현재 다양한 연구와 시도가 진행되고 있다. 이론 내용의 암기나 프로그래밍 언어에 학습에 치중한다면 학습자의 문제 해결 역량을 충분히 향상 시킬 수 없으므로, 컴퓨팅사고를 효과적으로 문제 풀이 과정에 적용할 수 있는 교과 내용을 개발하고 그 효과를 검증하는 것이 중요하다.

이를 위해 기존의 컴퓨팅사고 교과목의 교육 효과에 대한 선행 연구들을 분석하고 이론적 지식에 대한 평가 사례와 산출물을 분석하여 학습자의 지식을 평가하는 사례를 살펴보았다. 그러나 컴퓨팅사고는 이론적 지식 습득이나 프로그래밍 산출물에 국한되어 그 효과가 나타나는 것이 아니라 문제를 해결하는 전 과정에 적용되는 사고 과정이다.

이에 본 논문에서는 컴퓨팅사고 교육의 효과를 분석하기 위해 컴퓨팅 문제 해결의 전체 과정이라 할 수 있는 분석, 설계, 구현 및 테스트 단계에 따라 컴퓨팅사고 개념을 분류하고 평가 기준을 구성하였다. 제안 기준에 따라 전통적인 프로그래밍 언어 교육에 비교하여 컴퓨팅사고가 문제 해결 능력 향상에 어떠한 유의미한 차이를 가져오는지 전체 문제 해결 과정을 세분화하여 평가하였다.

컴퓨팅사고 교과목과 프로그래밍 언어 교과목을 각각 학습한 그룹의 문제 해결 과정의 결과를 평가한 결과 문제 분석 및 설계 과정에서 컴퓨팅사고 개념을 학습한 그룹이 더 체계적으로 문제를 해결하는 것을 알 수 있었다. 프로그래밍 언어 교과목을 수강한 그룹도 최종 결과물은 성공적으로 구현하였으나 문제의 분석과 설계 과정에 다소 미숙함을 보여주었으며, 문제의 복잡성이 증가할수록 컴퓨팅사고의 적용이 중요해질 것이다.

본 연구에서는 문제 해결 과정에서 컴퓨팅사고 개념과 프로그래밍 언어를 통한 표현 능력의 중요성에 대해 분석하였다. 향후 컴퓨팅사고 및 이를 활용한 다양한 응용 과목에 분석 결과를 참고하여 개발 방향을 제시할 수 있을 것이다.

REFERENCE

[1] H. Rim, “Computational Thinking, Algorithm, Creativity, Problem-solving Ability, Programming Education, Software Education, Code.org Site,” *Journal of Korea Multimedia Society*, Vol. 20, No. 2, pp. 382-395, 2017.

[2] H. Rim, “Android App. Implementation Teaching Using App. Inventor for Elementary School Students,” *Journal of Korea Multimedia Society*, Vol. 16, No. 12, pp. 1495-1507, 2013.

[3] J. Lee and H. Ha, “Verification of the Effectiveness of Teaching-learning Model for Maker Education Based on SW Coding,” *The Korean Society for Creative Information Culture*, Vol. 2, No. 2, pp. 49-61, 2016.

[4] M. Kim, G. Yoo, and H. Kim, “Development of a Scoring Rubric Based on Computational Thinking for Evaluating Students’ Computational Artifacts in Programming Course,” *The Korean Association of Computer Education*, Vol. 20, No. 2, pp. 1-11, 2017.

[5] J. Lee, “Exploration for Developing Assessment Tools for Computational Thinking,” *Journal of Creative Information Culture*, Vol. 4, No. 3, pp. 273-283, 2018.

[6] Y. Sun, “App Inventor Learning Model for SW Education in Elementary School,” *Journal of Creative Information Culture*, Vol. 2, No. 2, pp. 63-72, 2016.

[7] B. Rodriguez, S. Kennicutt, C. Rader, and T. Camp, “Assessing Computational Thinking in CS Unplugged Activities,” *Proceeding of the 2017 ACM Special Interest Group Computer Science Education Technical Symposium on Computer Science Education*, pp. 501-506. 2017.

[8] L. Gouws, K. Bradshaw, and P. Wentworth, “First Year Student Performance in a Test for Computational Thinking,” *Proceeding of the South African Institute for Computer Scientists and Information Technologists Conference*, pp. 271-277, 2013.

[9] H. Bort and D. Brylow, “CS4Impact: Measuring Computational Thinking Concepts Present in CS4HS Participant Lesson Plans,” *Proceeding of the 44th ACM Technical Symposium*

on Computer Science Education, pp. 427-432, 2013.

[10] S. Kim, S. Ham, and K. Song, "Analytic Study on the Effectiveness of Computational Thinking Based STEAM Program," *The Korean Association of Computer Education*, Vol. 18, No. 3, pp. 105-114, 2015.

[11] J. Kwon and J. Kim, "A Study on the Relationship between Computational Thinking Based SW Education and Problem Solving," *The Korean Association of Computer Education*, Vol. 21, No. 1, pp. 9-10, 2017.

[12] M. Romero, A. Lepage, and B. Lille, "Computational Thinking Development through Creative Programming in Higher Education," *International Journal of Educational Technology in Higher Education*, Vol. 14, No. 1, pp. 1-15, 2017.

[13] S. Kim, "Analysis of Scratch Code for Student Assessment about Computational Thinking Capability," *The Korean Association of Computer Education*, Vol. 18, No. 5, pp. 25-34, 2015.

[14] H. Choi, "Developing Lessons and Rubrics to Promote Computational Thinking," *Journal of The Korean Assocaition of Information Education*, Vol. 18, No. 1 pp. 57-64, 2014.

[15] G. Elliott and J. Strachan, *Global Business Information Technology*, Addison-Wesley Publishers, United States of America, 2004.

김 재 경

2007년 연세대학교, 컴퓨터과학과 박사
2007년~2009년: University of Pittsburgh, Post-Doc
2009년~2010년 연세대학교, 연구교수
2013년~2017년 연세대학교, 객원교수
2017년~현재 연세대학교 학부대학 조교수
관심분야 : 소프트웨어 교육