JDCS
디지털콘텐츠학회논문지
Journal of Digital Contents Society
Vol. 20, No. 12, pp. 2423-2432, Dec. 2019

비전공자를 위한 구조적 소프트웨어 교육 모형-파이썬 중심으로

강 의 선
숭실대학교 베어드교양대학

Structural Software Education Model for Non-majors - Focused on Python

Euisun Kang
Baird College of General Education, Soongsil University, Seoul 06978, Korea

[요 약]

최근 대학에서는 4차 산업혁명에 맞추어 디지털 융합 시대를 이끌어갈 인재를 육성하기 위하여 컴퓨팅적 사고를 기반으로 하는 소프트웨어 교육을 교과과목으로 편성 및 운영하고 있다. 하지만 비전공자에게 소프트웨어 교육은 새로운 개발 환경과 경험의 부재로 난관에 부딪히고 있다. 본 연구는 비전공자들이 소프트웨어를 학습하는 과정에서 느끼는 어려움을 파악하기 위하여 학습자의 특성을 이해하고 비전공자를 위한 소프트웨어 교육 모형을 제안하는 데 목적이 있다. 학습자의 특성을 이해하기 위하여 파이썬 프로그래밍 언어 교육을 진행하였고 프로그래밍 학습의 어려움과 컴퓨팅적 사고와의 관련성을 조사하였다. 그 결과 문법에 대한 이해 부족, 절차적 문제해결에 따른 사고력의 차이 등으로 학습의 어려움을 느끼고 있음을 확인하였으며 이를 극복하기 위한 컴퓨팅적 사고 기반의 구조적 소프트웨어 교육 모형을 제안하고자 한다.

[Abstract]

Recently, in order to foster talents who will lead the digital convergence era in line with the 4th Industrial Revolution, the university organizes and operates software education based on computing thinking as a curriculum. But for non-technologists, software education is facing challenges due to the lack of new development environments and experience. The purpose of this study is to understand the characteristics of learners and to suggest a software education model for non-majors in order to understand the difficulties that non-majors experience in learning software. In order to understand the characteristics of learners, we conducted a Python programming language education and examined the relationship between the difficulty of programming learning and computing thinking. As a result, we confirmed that non-majors felt difficulty in learning due to lack of understanding of grammar and differences in thinking ability due to procedural problem solving, and proposed a structural software education model based on computing thinking to overcome this problem.

색인어 : 컴퓨팅적 사고, 교육 모형, 문제해결, 파이썬, 소프트웨어 교육

Key word : Computational Thinking, Education Model, Problem Solving, Python, Software Education

http://dx.doi.org/10.9728/dcs.2019.20.12.2423

Received 11 November 2019; **Revised** 30 November 2019
Accepted 15 December 2019

***Corresponding Author; Euisun Kang**

Tel: +82-2-828-7264
E-mail: iami86@ssu.ac.kr

Ⅰ. 서 론

최근 정보통신 기술의 발달로 정보화 혁명을 이루었던 산업 분야는 소프트웨어의 확산과 기술 간의 융합을 기반으로 새로운 디지털 시대로 산업 흐름의 변화를 보여주고 있다. 4차 산업 혁명을 살아가는 디지털 시대는 모든 영역에서의 융합을 가속화하는 소프트웨어를 벗어나긴 힘들 것이다. 따라서 각 분야의 경험과 아이디어를 소프트웨어라는 형태로 공유하기 위해서는 컴퓨터 과학 원리를 적용하여 현실 세계의 문제를 해결할 수 있는 컴퓨팅적 사고 능력이 필요하다[1]. 컴퓨팅적 사고는 문제를 해결하기 위한 접근방법으로써 전공과 관계없이 디지털 시대를 살아가는 누구나 배워야 하는 사고력이다[2]. 국내외 교육기관에서 컴퓨팅 사고력을 함양하기 위해 편성 및 운영하고 있는 교육으로는 저학년을 위한 언플러그드 교육을 포함하여 블록기반 프로그래밍과 고급 프로그래밍 언어가 있다. 컴퓨터 이론 수업에 대한 문제점을 극복하기 위한 고등 교육기관에서 이루어지는 고급 프로그래밍 언어 중심의 소프트웨어 교육은 문제를 절차적으로 해결하는 과정을 거침으로써 컴퓨터 원리를 이해하는데 많은 효과성을 보여주고 있다. 하지만 인문, 사회, 문화의 전반적인 문제를 학습한 대학의 비전공자에게는 프로그램 작성 환경의 경험 부족, 절차적 문제해결에 따른 사고력의 차이, 전공과의 연관성 등으로 학습의 어려움을 느끼고 있다. 특히 시각적으로 프로그래밍 작성과정을 보여주는 블록기반과는 다르게 텍스트 기반의 고급 프로그래밍 학습은 생소한 문법 습득의 어려움, 실행과정에서 발생하는 오류 등으로 인해 소프트웨어의 지적 호기심을 저해하는 요소로 작용하고 있다[3][4][5]. 이는 비전공자들이 학습하였던 사고력과 컴퓨팅 사고력의 차이에서 오는 결과라 할 수 있다[6][7].

본 연구에서는 비전공자들이 컴퓨팅 사고력을 함양하기 위한 소프트웨어 교육 과정에서 발생하는 문제점을 이해하기 위해 고급 프로그래밍 언어 교육에서 비전공자들의 학습 특성을 도구의 사용, 발생하는 오류의 어려움, 문법의 이해정도와 컴퓨팅 사고력과의 관련성 측면으로 사후 설문을 시행하여 분석하고 이를 극복하기 위한 컴퓨팅 사고력 기반의 구조적 소프트웨어 교육 모형을 제안하고자 한다.

Ⅱ. 관련연구

2-1 컴퓨팅적 사고

2006년 Jeannette Wing은 컴퓨팅적 사고를 문제 해결을 위한 정보를 처리하는 사람이나 컴퓨터가 효율적으로 처리할 수 있는 형태의 문제해결과정이라고 정의하였다[2]. 컴퓨팅적 사고의 중요성을 인식한 우리나라의 경우 2015년 정보과 교과과정을 개정하면서 소프트웨어의 운영지침으로 컴퓨팅적 사고를 컴퓨팅의 기본적인 개념과 원리를 기반으로 문제를 효율적으로 해결할 수 있는 사고 능력으로 정의하고 있다[9]. 정리하면 컴퓨팅적 사고는 과학의 원리와 개념을 바탕으로 문제를 해결하는 절차적이고 체계적인 과정이다. 컴퓨팅적 사고를 학습하기 위한 세부 구성요소를 정리하면 표 1과 같다.

표 1. 컴퓨팅적 사고의 구성요소[3][9][10]
Table 1. Component of Computational Thinking[3][9][10]

Year/Researcher	Component
CSTA&ISTE (2011) [10]	Data Collection Data Analysis Data Representation Problem decomposition Abstraction Algorithm&Procedure Automation Simulation Parallelization
Google, England BBC Computing Education (2016)	Decomposition Pattern Recognition Abstraction Algorithm Design
Korean Education Development Institute (2015)[9]	Decomposition Pattern Recognition Abstraction Algorithm Programming

CSTA(Computer Science Teachers Association)는 ISTE(International Society for Technology in Education)와 공동으로 K-12 학습자에게 체계적으로 컴퓨터 과학을 교육할 수 있는 교육 과정 기준안으로 구성요소를 9가지로 구성하였다[3]. 교사들은 이 과정을 통하여 학생들이 문제를 규정하고 타인과의 협업을 통하여 해법을 찾는 능력을 배양하도록 하고 있다. Google은 영국의 BBC 컴퓨팅 교육에서 제안하는 분해, 패턴인식, 추상화, 알고리즘 설계로 제안하고 있다. 2015년 국내 한국교육개발원의 정책 연구에서는 소프트웨어 교육을 위한 컴퓨팅 사고력의 학습 모델로서 분해, 패턴인식, 추상화, 알고리즘의 4가지 항목과 선택적 프로그래밍 항목을 제시하였다[9].

2-2 대학 교양 교육에서 컴퓨팅적 사고 교육

2016년 교육부는 컴퓨팅적 사고 중심의 소프트웨어 교육을 대학으로 확대하기 위하여 소프트웨어 육성 사업을 제안하고 2019년까지 약 35개 대학을 선정하여 운영하고 있다. 이와 관련하여 필요성을 느낀 대학들은 전교생을 대상으로 컴퓨팅 사고력 함양을 위한 소프트웨어 교육을 교양 필수 교과목으로 편성 및 운영하고 있으며 컴퓨터 전공자를 포함하여 비전공자도 미래의 4차 산업 혁명의 핵심 기술인 사물 인터넷, 인공지능, 빅 데이터등과 같은 창의·융합 사고력을 키우기 위해 노력하고 있다.

[11]은 컴퓨팅적 사고와 소프트웨어 교육을 교양필수 교과목으로 신설하기 전에 과목 개설에 대한 사전 설문에서 소프트웨어 과목 개설시 수강 의향에 대한 응답이 65.3%로 많았음을 확인하였다. 그 이유로는 전공 분야와의 융합, 취업에 유리한

요소로 작용, 창업에 활용, 취미 및 개인의 흥미, 다양한 활용성 순으로 나타났다. 그러면서 컴퓨팅적 사고를 교양 과목으로 운영시 컴퓨터 개론과 같은 암기식 교육이나 획일적인 단순 코딩 교육을 지양하고 컴퓨팅 사고를 통한 문제 해결 능력과 기술을 교육하여 다양한 문제에 적용이 가능하도록 기반을 마련해 줄 것을 당부하였다.

[12]는 3년 동안 컴퓨팅적 사고를 교양 교과목으로 운영하는 과정에서 학생들의 교과목 인식 추이에 대한 분석 결과를 제시하였다. 교양필수로서의 적합성 인식 분석은 긍정적인 결과를 보였고 도입 초기에 비해 운영 기간이 지날수록 교양필수로의 적합성이 더 향상됨을 확인하였다. 교과목 효과성 부분에서는 '전공과의 융합 유용성'이 '취업 및 진로 유용성'보다 높은 결과를 확인할 수 있었다. 소프트웨어 교육 측면에서는 텍스트 기반의 고급 프로그래밍 언어인 파이썬이 다소 어렵다고 느껴지는 성향이 있었지만 기초 소프트웨어 교육으로써 적합하다는 인식 결과를 확인할 수 있었다. 수강 학년에 따라 다소 차이가 있지만 전반적으로 학습자 스스로 4차 산업 혁명 변화와 도전에 대한 관심에 맞추어 대학에서 갖추어야 할 역량으로 컴퓨팅적 사고에 대한 필요성이 높음을 확인할 수 있었다.

2-3 교양 교육으로의 소프트웨어 학습에서 고려 사항

컴퓨팅적 사고를 함양하기 위한 교육으로는 암기식 이론 수업에 대한 부정적인 측면을 극복하기 위하여 실습이 병합된 수업으로 대부분 진행되며 그에 대한 도구로써 소프트웨어 교육이 이루어지고 있다[11][12]. 대학에서 비전공자를 위한 소프트웨어 실습 교육은 크게 블록 기반 프로그래밍과 텍스트 기반 프로그래밍으로 나뉜다. 블록 기반 프로그래밍 수업은 스크래치(Scratch)와 앱인벤터(App Inventor)와 같은 시각적인 도구를 활용하여 학습하는 방법으로 프로그램의 개념, 구조와 흐름을 쉽게 이해할 수 있는 장점이 있다. 블록 기반의 교육용 프로그래밍 언어를 교육에 활용했을 경우 재미있는 캐릭터와 쉬운 인터페이스의 영향으로 외부 요인에 상관없이 학습이 원활하게 진행되며 이에 따라 학습자의 문제해결력, 논리적 사고가 상승되는 효과가 있었다[13][14][15][16]. 이와 더불어 자기 주도적 학습이 가능한 플립러닝 방식과 전문 지식없이 참여할 수 있는 경진대회는 코딩에 대한 자신감과 관심, 흥미도, 성취도가 함께 상승됨을 확인하였다[14]. 고급 프로그래밍은 텍스트 중심의 문법을 이용하여 프로그램의 흐름을 기술하는 방법으로 전문가들이 소프트웨어를 개발할 때 사용하는 방법이다. 교양교육에서는 비전공자임을 감안하여 개발 전문 프로그램 언어가 아닌 문법이 쉬운 파이썬과 같은 교육 프로그램으로 수업이 진행되고 있다. [17]은 파이썬 기반으로의 컴퓨팅 사고력 핵심 요소를 개발 중심 모델과 연계한 교육 모델을 제시하였고 결과에서 규칙과 추상화를 포함한 컴퓨팅 사고력 향상에 효과가 있음을 확인하였다.

위와 같이 비전공자를 위한 소프트웨어 교육에 관한 연구는 긍정적 효과에도 불구하고 발생하는 문제들을 극복하기 위해 많은 연구가 계속 진행되어야 함을 보여주고 있다[13][15]. 비전공자를 고려한 블록 기반 프로그래밍의 결과에서 프로그래밍 어려움의 결과는 소프트웨어 교육의 이해 부족와 필요성 인식 부족의 결과를 초래하고 교양 필수 교과목에 대한 학점 부담으로 이어지고 있음을 보여주고 있다[13]. 학습자의 성별과 프로그래밍 과목에 따른 수강생의 학습이탈동기에 대하여 분석한 연구에서 스크래치 학습에 대해서는 성별에 대한 학습이탈동기에 차이가 없음을 확인하였고 파이썬 학습의 경우 남학생이 프로그래밍에 대한 자신감이 높아짐으로써 학습이탈동기가 개선되었음을 확인하였다[15]. [13][19]는 소프트웨어 교육이 학습자의 전공에 따라 관심도에 영향이 있음을 확인하였다. [19]는 표 2와 같이 전공계열을 크게 6개의 세부 그룹으로 분류하고 각 그룹에 대한 계열별 기초 소프트웨어 과목, 전공과의 융합에 대한 인식정도와 프로그래밍 도구 및 난이도를 그룹별로 조사하였다. 그리고 동일한 소프트웨어 교육이더라도 계열별 특성을 고려하여 교육 방향을 다르게 설계해야 함을 강조하였다.

표 2. 기초 소프트웨어 교육을 위한 6계열 그룹[19]
Table 2. Six Groups for Basic Software Education[19]

Major Group	Belonged Colleges
IT	College of IT
non-IT in SE	College of Engineering College of Natural Science
Humanities & Social Science	College of Humanites College of Law College of Social Science Dept. of Sports Dept. of Art Creation Dept. of Free major
Economics & Business	College of Business College of Economics& Trade
Free major	Dept. of Free major
Arts & Sports	Dept. of Sports Dept. of Arts Creation

Ⅲ. 텍스트 기반 소프트웨어 교육에서 비전공 학습자의 특성

정보처리 관점에서 문제 해결 사고력을 향상시키는 실습수업으로 프로그래밍 경험을 바탕으로 이루어지는 경우가 많다. 학습자의 특성을 고려한 교육용 프로그래밍 언어는 학습자의 관심과 흥미, 프로그램의 확장성을 고려하여 선택하는 것이 바람직하다. 파이썬(Python)은 1991년 귀도 반 로섬(Guido van Rossum)이 발표한 고급 프로그래밍 언어로써 인터프린터 방식의 쉬운 문법과 유용한 오픈소스 라이브러리를 제공하는 등 다양한 장점으로 컴퓨터 전공이 개설된 대학에서 채택하는 대표적인 프로그래밍 언어이다[18]. 하지만 교육용 프로그래밍 언어임에도 불구하고 비전공자에게 텍스트 기반의 파이썬 프로그래밍 언어는 새로운 개념과 문법이라는 진입장벽으로 인해 학습의 어려움을 느끼고 있다[17].

본 연구에서는 파이썬 중심의 소프트웨어 교육에서 학습자들이 어떤 부분에 어려움을 느끼는지 알아보고자 학습자의 특성을 조사해 보았다. A대학은 필수 교양 교과목인 "컴퓨팅적 사고"에서 컴퓨팅 사고력 함양을 목표로 파이썬을 이용한 소프트웨어 교육을 1학년 학생을 대상으로 실시하고 있다. 본 연구를 위하여 2019년 1학기 "컴퓨팅적 사고" 수업에 참여한 학생을 대상으로 표 4의 내용을 기반으로 수업을 진행한 후 설문을 실시하였다. 설문에 참여한 학생은 전체 63명이며 학습자의 학과(부)별 인원수는 표 3과 같다. 설문에 참여한 63명의 학생은 프로그램의 경험이 없는 컴퓨터 비전공자였으며 공과대학과 자연과학대학에 소속학과로 편성되어 있다.

표 3. 응답자 소속 학과
Table 3. Major profile of respondents

Major	Distribution
School of Architecture	20
Department of Organic Materials and Fiber Engineering	11
Statistics and Actuarial Science	12
Department of Chemistry	20
Total	63

표 4. 수업내용
Table 4. Syllabus

week	On-line	Off-line
1	- Definition of Computing Thinking - Components of Computing Thinking	- Definition of Computing Thinking - Components of Computing Thinking - Python Introduction
2	- Algorithm Definition - Algorithm Representation Method - Variable, Operator, List	- Algorithm Representation Method - Variable, Operator, List
3	- print() / input() - Conditional statement	- Print() practice - Input() practice
4	- Loop - Use of loops - Turtle Library	- Midterm exam - Conditional statement practice
5	- Function Definition - Function practice	- Loop - Loop practice - Turtle practice
6	- Solve the problem recursively	- Function Definition - Function practice
7		Final exam

수업은 7주간 온라인 50분, 오프라인 100분으로 이루어진 블랜디드 러닝(Blended Learning) 수업으로 진행하였다. 온라인 수업에서는 컴퓨팅적 사고의 이론과 파이썬 프로그래밍의 이론 및 실습내용을 학습하였고 오프라인 수업에서 파이썬 프로그램 중심의 실습을 통하여 컴퓨팅적 사고의 구성요소와 정보처리 관점의 문제 해결 과정을 학습하였다. 매 수업 연습문제를 통하여 파이썬을 이용한 일상적인 문제들을 프로그램으로 해결하는 학습을 진행하였다. 수업이 완료된 후 컴퓨팅적 사고와 관련하여 프로그래밍이 어려운 이유를 분석하기 위해 프로그래밍이 어려운 이유와 컴퓨팅적 사고와의 관계를 기술한 [7]을 활용하여 사후 설문을 실시하였고 학생들이 느끼는 프로그램 작성 단계의 어려움을 확인하여 보았다.

표 5. 파이썬 도구의 사용과 오류 메시지의 해결 정도
Tabel 5. Use of Python Tools and the Resolution of Error Messages

	Very Easy	Easy	Normal	Diff	Very Diff
The language of the tool	0.0%	22.2%	34.9%	23.8%	11.1%
Tool usage	0.0%	23.8%	31.7%	25.4%	14.3%
Language of the error	0.0%	15.9%	39.7%	30.2%	9.5%
Understanding the error	0.0%	19.0%	31.7%	36.5%	11.1%
Error correction	0.0%	14.3%	36.5%	34.9%	12.7%

표 5는 파이썬을 작성하는 도구의 사용과 작성한 코드를 실행하는 과정에서 발생하는 오류 메시지를 해결하는 과정의 어려움을 분석한 것이다. 디지털 환경에 노출된 대학생들은 파이썬을 작성하는 언어 및 사용에 대해 어려움은 크게 없었다. 하지만 프로그램의 정상적인 실행을 위해 발생하는 오류를 이해하고 수정하는 부분에서는 보통을 제외하면 50% 이하이지만 어려움을 느끼는 학생들이 있었다. 이는 문법에 대한 사용법과 이해 부분에서 오류에 대한 다양한 경험을 접하지 못한 결과라 할 수 있으며 오류의 원인이 문법이나 명령어의 어떤 부분에서 발생하는지 모르거나 오류의 의미는 이해가 되나 어느 부분을 수정해야 하는지 모르는 경우이다. 또한, 수정한 오류 결과의 영향으로 다른 오류가 발생하는 경우 대처하는 방법이 미흡함에서 알 수 있듯이 정확한 파이썬 문법의 사용과 이해에 영향이 있음을 알 수 있었다.

표. 6. 파이썬 문법의 이해도 정도
Tabel 6. Understanding of Python syntax

	Very Easy	Easy	Normal	Diff	Very Diff
Create Variables	23.8%	19.0%	42.9%	9.5%	4.8%
Using Variables	20.6%	22.2%	39.7%	12.7%	4.8%
Create List	7.9%	23.8%	39.7%	23.8%	4.8%
Using List	6.3%	7.9%	44.4%	30.2%	11.1%
Print	33.3%	33.3%	23.8%	4.8%	4.8%
Input	28.6%	30.2%	25.4%	11.1%	4.8%
Using conditional	9.5%	25.4%	31.7%	20.6%	12.7%
Using loops	3.2%	9.5%	36.5%	28.6%	22.2%
Defining Functions	3.2%	7.9%	25.4%	33.3%	30.2%
Using Functions	3.2%	4.8%	27.0%	36.5%	28.6%

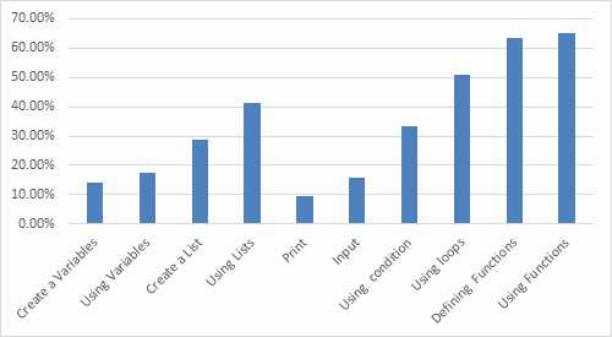

그림 1. 파이썬 문법에 따른 어려움 정도
Fig. 1. Degree of difficulty with Python syntax

수업에서는 학생들이 주어진 일상 문제를 파이썬을 이용하여 자동화하는 과정에서 문법적으로 어떤 어려움을 느끼는지 알아보기 위해 기본적인 파이썬 문법을 기반으로 수업을 진행하였다. 수업에서 학습한 문법은 변수, 1차원 리스트, 입출력문, 조건문, 반복문, 함수이다. 이 과정에서 학생들이 느끼는 각 문법의 이해도 및 어려움에 대해 표 6와 그림 1과 같은 결과를 얻을 수 있었다. 문제를 자동화하는 과정에서 사용자로부터 입력을 받거나 결과를 출력하는 부분과 분석한 문제를 기반으로 변수를 생성하고 사용하는 부분에서는 큰 어려움이 없었다. 하지만 변수의 배치를 바꾸거나 변수에 저장된 값이 다음 프로그래밍에 영향을 준다는 사실을 깨닫지 못하는 경우가 발견되었다. 이는 알고리즘 처리 과정 중 문제 해결을 자동화하는 데 필요한 명령들을 구체적인 순서에 따라 나열하는 순차처리에 대한 이해 부족과 변수에 대한 정확한 이해 부족으로 해석할 수 있었다[23]. 리스트의 경우 여러 개의 값을 저장하기 위하여 리스트를 생성하는 것은 어려움이 없어 보이나 첨자(index)를 이용한 리스트의 사용은 어려움을 느끼는 것을 확인할 수 있었다[4]. 리스트를 선언함과 동시에 값을 초기화 및 저장하거나 값을 변경하는 경우에는 리스트의 사용이 수월하였지만 비어있는 리스트에 특정 값을 삽입하거나 삭제, 추가하는 경우에는 리스트 문법을 잘못 사용하거나 문제를 해결하지 못하는 경우가 많았다. 특히 리스트를 효율적으로 다루기 위해서는 반복문을 활용하는 경우가 많은데 반복문 사용에 대한 어려움이 50.8%임을 감안한다면 리스트 사용 역시 어렵게 느껴질 수밖에 없다. 표 6의 결과를 차트로 표현한 그림 1에서도 확인할 수 있듯이 조건문, 반복문, 함수 부분으로 갈수록 어려움을 느끼는 정도가 많았다. 알고리즘 처리에서 선택처리는 프로그램을 논리적인 비교에 의해 특정 동작을 수행하도록 분기를 하는 것이고 반복처리는 문제분석 과정에서 반복적으로 실행하는 부분을 간략하게 표현하는 것이다[20]. 연습문제를 통하여 학습자의 오류 유형을 확인한 결과 문제에서 조건을 찾아 해당 문법으로 작성하는 것은 능숙하지만 중첩 조건에서는 조건식의 순서 및 포함 관계를 잘못 기재하거나 관계 연산자 또는 논리연산자를 잘못 작성하는 경우가 많았다. 특히 조건문과 반복문의 경우 각 조건 및 반복 횟수에 따라 수행해야할 명령어의 잘못된 포함 관계에 의해 문법적 흐름 오류가 발생하는 경우를 확인할 수 있었다[5]. 함수의 경우 함수를 정의하고 사용하는 부분 모두 어려움을 호소하는 횟수가 크게 증가하였다. 함수 또는 이벤트는 다른 동작을 위해 실행 제어권을 넘기는 개념이다. 학습자의 특성을 관찰한 결과 함수에 대한 필요성과 함수 호출 , 매개변수 사용, 반환값등 함수 정의 및 사용 방법에 대한 이해 부족으로 함수를 사용하여 프로그램을 독립적으로 분리하기 보다는 순차처리나 반복문을 사용하여 문제를 해결하려는 경향이 두드러지게 나타났다. 연습문제를 통하여 해결과정을 관찰한 결과 다수의 학습자들은 새로운 문제가 주어졌을 때 예제 코드를 바탕으로 최대한 같은 흐름과 방식으로 문제를 해결하려는 경향을 보였고 새로운 흐름의 문제인 경우에는 이미 학습한 명령어임에도 불구하고 명령어 선택에 주저하는 모습을 보였다[3][21]. 이와 같은 결과는 문제해결과정에서 문법을 연결하여 생각하기보다는 개별의 문법에 집중하는 모습의 결과이며 컴퓨터에 대한 이해 부족과 문제해결과정의 흐름을 기술하지 못하는 문제로 이어짐을 확인할 수 있었다.

[7][8]은 대학에서 초보 학습자들이 텍스트와 스크립트 기반 언어 중심의 프로그램 학습단계에서 보이는 어려움의 정도를 컴퓨팅 사고 능력간의 관계성으로 분석하였고 그 결과 프로그래밍을 어려워하는 사고 영역들이 컴퓨팅 사고력의 9가지 사고[10]와 관계가 있음을 확인하였다. 이를 파이썬 기반의 소프트웨어 교육과 컴퓨팅적 사고와의 관계성을 알아보기 위하여 [7]을 기반으로 조사해 보았다. 그 결과 동일하게 문제이해 단계, 문제 분석 단계, 알고리즘 사고 단계의 모든 단계에서 어려움을 느끼는 것을 표 7과 같이 확인할 수 있었다[7][8].

표 7. 컴퓨팅적 사고 요소에서 바라본 어려움
Tabel 7. Difficulties in Computational Thinking Elements

	Very Easy	Easy	Normal	Diff	Very Diff
Understanding the problem	0.0%	20.6%	33.3%	28.6%	17.5%
Data analysis	0.0%	12.7%	39.7%	30.2%	15.9%
Data Representation & Abstract	0.0%	9.5%	30.2%	41.3%	19.0%
Problem Decomposition	0.0%	12.7%	34.9%	28.6%	23.8%
Abstract	0.0%	14.3%	33.3%	33.3%	19.0%
Algorithm	0.0%	9.5%	34.9%	33.3%	22.2%
Automation	0.0%	11.1%	34.9%	31.7%	22.2%
Simulation	0.0%	9.5%	36.5%	36.5%	17.5%

표 7에서 볼 수 있듯이 주어진 문제를 해결하고자 하는 것이 무엇인지에 대한 문제 이해 및 정의 부분에 대하여 다른 항목에 비해 어렵다는 결과가 적게 나왔다. 문제 이해에 대한 방법들을 확인한 결과 해결해야 하는 문제에 대한 가상의 결과 예를 스스로 설정함으로써 문제에 대한 정확한 이해에 접근하는 것을 알 수 있었다. 하지만 결과를 도출하기 위하여 문제에서

제시하는 정보가 무엇인지 파악하는 자료 분석은 50%이상으로 다소 높았고 이는 데이터 표현, 추상화 그리고 분해와 관련이 있었다. 데이터 표현과 추상화 부분은 60.3%, 52.3%로 어려워하는 부분으로 분석되었다. Wing은 추상화와 알고리즘을 컴퓨팅적 사고 구성 요소 중 가장 중요한 요소로 정의하였다[2]. 추상화는 복잡한 문제를 단순화하여 문제를 일반화시키고 세부사항들을 분리하여 구체적으로 기술함으로써 단순화시키는 것을 포함한다. 데이터 분석에서 정확한 자료 검출은 문제의 일반화와 연결될 수 있다[22]. 이를 프로그래밍 문법과 연결한다면 변수(리스트) 생성과 사용, 입력 데이터 설정, 연산식 도출, 연산자 선택, 문법 선택, 함수의 사용 여부, 처리 과정에서의 자료 선택 등에 대한 분석이 미흡한 결과라 할 수 있다. 문제 분해에서는 52.5%의 어려움에 대한 답변이 있었다. 학습자들은 주어진 문제를 작은 단위의 문제로 분해하는 이론적 접근은 이해하지만 알고리즘과 자동화 단계로 이어지는 경우 앞에서 보여진 결과에서처럼 프로그램을 작은 단위로 묶어서 처리해야 하는 필요성과 이해의 부족으로 어려움을 호소하는 경향이 있었다. 학습자는 알고리즘, 자동화, 시뮬레이션 부분에서도 어려움(보통 이상)을 느끼고 있었다. 문제 해결 절차를 정보처리관점에서 문제를 해결하기 위해서는 논리적이고 체계적인 순서로 과정을 기술해야 자동화 단계로의 연결이 쉬워진다. 컴퓨팅적 사고는 논리적으로 분석된 자료를 추상화를 통해 자료와 문제를 표현하고 공식화하여 컴퓨터 과학 원리를 이용하고 절차적 사고를 통하여 해결과정을 자동화하는 일련의 사고이다[9]. 하지만 결과에서 볼 수 있듯이 데이터 분석과 데이터 표현 그리고 추상화 단계의 이해 부족은 정보처리 관점에서 기술되어야 하는 알고리즘 설계에도 영향을 미치는 것으로 볼 수 있었다. 순차처리, 선택처리, 반복처리를 기반으로 하는 알고리즘의 이해는 자동화로 이어지는 중간단계이다. 따라서 잘못된 알고리즘 설계는 자동화로 옮겼을 때 논리적 오류 뿐만 아니라 흐름 오류를 발생시킬 수 있다[23].

앞에서 살펴본 바와 같이 컴퓨팅적 사고와 프로그래밍의 어려움에 대한 관계를 종합적으로 볼 때 비전공자는 문제를 소프트웨어로 변환하는 과정에서 파이썬의 문법에 대한 어려움 뿐만 아니라 컴퓨팅적 사고 요소에 따른 모든 단계에서 어려움을 느끼는 것을 확인하였다[8]. 컴퓨터의 원리를 활용하는 텍스트 기반의 고급 프로그래밍 문법 구조는 엄격하다. 프로그램의 흐름에 대한 이해 보다는 문법 자체에 집중함으로써 비전공 학습자들에게 흥미 저하 및 인지적인 부담감을 증대시킬 수 있다. 따라서 학습 초기 단계부터 프로그램의 학습 도입은 적절하지 않을 수 있으므로 프로그래밍 언어의 구문과 의미등 프로그래밍 지식보다 프로그램을 작성하는 설계 단계의 프로그램 전략이 선행되어야 할 것이다[24]. 이를 위해 문제 이해부터 자동화까지 모든 단계에서 프로그래밍을 어려워하는 학생들에게는 인지적 부담을 해소하기 위하여 프로그래밍 기반의 문제 해결 과정 학습에 집중할 수 있는 사전 교육이 필요하며 각 문법의 연관 관계를 이해하고 활용성을 높이기 위해서는 단계별 문법 중심의 수업보다는 반복적 문법 학습에 집중할 필요가 있다. 본 연구는 비전공 학습자를 위한 컴퓨팅 사고력 기반의 소프트웨어 교육 학습 요건을 다음과 같이 정리하였다.

표 8. 소프트웨어 교육의 학습 요건[5][23][25]
Table 8. Learning Requirements for Software Training [5][23][25]

Item	Contents
Easy Grammar Focus	Easy grammatical keyword-based learning
small size	Constructed with little grammar and meaning that you can easily learn in a short time
core concept	The core concepts are presented accurately and consistently, and the programming proceeds only with the suggested core concepts.
connection	Transition to the area of programming language based on common problems and design
Design-based learning	Design-driven learning to understand the overall flow of the program

IV. 컴퓨팅 사고력을 고려한 구조적 프로그래밍 교육 모형

4-1 제안하는 학습모형에 대한 학습 목적

본 연구에서 제안하는 교육모형에 대한 학습 목적은 다음과 같다.

첫째, 컴퓨팅적 사고의 개념을 이해하여 프로그래밍 언어로 연계가 가능하도록 한다.

컴퓨팅적 사고는 일상의 문제를 정보처리관점에서 해결하는 사고력이다. 주어진 문제를 정보처리관점에서 해결을 위해서는 주어진 문제를 분석하고 처리하며 자동화 과정을 통하여 결과를 확인할 수 있어야 한다. 이를 위하여 컴퓨팅적 사고에 대한 이해, 각 구성요소들에 대한 이해가 필요하다. 일상의 문제를 컴퓨팅적 사고의 구성요소로 분석함으로써 전반적인 문제 해결 과정에 집중할 수 있도록 하고 자동화 과정을 위한 준비 단계가 되도록 한다.

둘째, 일상의 문제를 프로그래밍 언어를 이용하여 재구성함으로써 컴퓨터에 대한 이해를 돕는다.

컴퓨터의 유저(User)입장에 익숙한 비전공자 학습자에게는 물리적인 컴퓨터에 대한 이해가 어렵게 느껴질 수 있으므로 소프트웨어 교육을 통하여 컴퓨터의 문제 처리 과정을 직접 경험할 수 있도록 한다. 프로그램 단계에서 발생하는 명령어 기술, 프로그램 처리의 흐름 이해, 발생하는 오류들을 직접 경험함으로써 비전공자 학습자가 사용하는 소프트웨어에 대한 이해를 도울수 있을 뿐만 아니라 컴퓨터에 대한 전반적인 이해에 도움을 줄 수 있도록 한다.

셋째, 고급 프로그래밍 언어 작성에 대한 학습자의 어려움을 이해하고 쉽게 작성할 수 있도록 한다.

프로그래밍이란 문제 영역을 프로그래밍 영역으로 변환하는 과정을 의미한다. 텍스트를 이용하여 명령어를 입력하여 흐름을 기술하는 고급 프로그래밍 언어 교육은 프로그램 이해보다는 프로그램 작성에 중심을 두기 때문에 획일적인 명령어 문법 중심으로 교육이 이루어지고 있으며 문법의 학습 깊이에 따라 학습자는 소프트웨어 학습에서 느끼는 어려움을 다르게 느끼고 있다[25]. 따라서 비전공자를 위한 소프트웨어 교육에서는 핵심적인 중심 문법과 지속적인 반복 학습을 기반으로 문법에 대한 이해와 활용 방법을 익혀 프로그램을 이해하고 작성할 수 있도록 한다.

넷째, 구조화된 소프트웨어 설계를 통하여 문제에 대한 전반적인 흐름을 파악하여 복잡한 문제를 해결할 수 있는 역량을 키운다.

문법 기반 중심으로 학습한 초보 프로그래머들은 변수, 입출력, 제어문에 대한 각 명령어들이 무엇인지는 알고 있지만 이를 문제 상황에 적용하지 못하여 프로그래밍이 어렵게 느껴지고 흥미가 저하되었다[25]. 문법 중심의 소프트웨어 교육은 주어진 문제에 관한 결과를 도출해 내는 전체 흐름에 대한 이해를 어렵게 한다. 좀 더 나아가 모듈과 라이브러리를 능숙하게 확장하여 활용하기 위해서는 결국 프로그램 흐름(과정)에 대한 기본적인 사고력인 알고리즘 설계가 우선 학습되어야 한다. 알고리즘을 표현하는 방법으로는 자연어, 순서도, 의사코드가 있다. 순서도는 특정 기호 및 도형을 사용하여 분석된 문제를 단계적인 순서로 기술하여 상호 간의 관계 흐름을 표현한 다이어그램 기반 학습 방법이다[26]. [27]은 순서도 학습이 언어의 복잡도에 대한 간결성과 관련있는 여러 개념을 하나의 개념으로 통합하는 데 유용하므로 일반성면에서 블록기반 프로그래밍 보다 우세함을 확인하였다. 순서도는 도형을 사용하여 문제해결 과정을 기술함으로써 문법을 학습하거나 문법에 대한 부담을 주지 않는다. 또한 각 도형은 문법적 의미를 내포하므로 도형별 고급 프로그래밍 문법으로의 쉽게 변환이 가능하다. 따라서 특정 문법만을 집중적으로 바라보는 습관을 줄일 수 있으며 문제해결 관점에서 프로그램의 흐름을 바라볼 수 있는 장점이 있으므로 프로그래밍을 접하기 전에 순서도를 이용한 설계 과정을 거침으로써 결과에 따른 프로그램의 흐름을 이해하는 데 도움을 줄 수 있다.

4-2 구조적 프로그래밍 교육 모형

본 연구에서는 소프트웨어 교육 모델의 형태와 학습 전략 수립에 따라 컴퓨팅 사고력 핵심 요소와 개발중심(DDD) 모델을 연계하여 비전공자를 위한 구조적 소프트웨어 교육모델을 제안하고자 한다. 제안한 모델의 개요는 표 9과 같다[13][15].

개발중심모델은 소프트웨어 공학의 전략 중 폭포수 모델을 바탕으로 요구분석-> 설계->구현->시험 과정의 Top-Down 방식의 개발절차를 활용한 모델로써 각 단계를 명확히 구분하여 명세화하는 특징을 갖고 있다[28].

표 9. 구조적 프로그래밍 교육 모형에서 DDD 모델 개요[13][15]
Table 9. DDD model overview in structured programming education model[13][15]

Step	Learning method
Discovery	Problem Analysis based on Computing Thinking (Problem Decomposition, Pattern recognition, Abstraction)
Design	Algorithm Planning and Design Using Flowchart
Development	Implementation and feedback in programming languages

탐구 단계에서는 학습자의 선행 지식을 활용하여 주어진 문제의 의미를 해석하고 이해하는 단계이다. 이 단계에서 파악할 수 있는 내용으로는 자료의 정의, 문제의 분해, 반복적인 규칙을 발견하여 문제를 일반화하고 추상화할 수 있는 핵심 내용을 파악하는 데 있다. 분석한 자료들을 토대로 변수 설계, 입력/출력 가정, 제어 구조(순차, 선택, 반복, 분해)의 흐름을 파악한다. 이로써 컴퓨팅적 사고의 구성요소에서 문제 분석, 분해, 패턴 인식, 추상화 단계를 학습하고 구체화할 수 있다.

설계 단계에서는 탐색 단계에서 분석된 자료 및 요소들을 순서도를 사용하여 문제해결과정을 기술하는 단계이다. 이는 컴퓨팅적 사고 요소 중 알고리즘에 해당하는 부분으로써 분석된 자료와 데이터를 중심으로 알고리즘의 처리 과정인 순차처리, 선택처리, 반복처리를 도형을 사용하여 구체적으로 기술하는 과정이다. 이 과정에서 기술되어야 하는 사항들은 변수 설정, 연산자 선택 및 기술, 입력/출력 정의, 문제 해결 흐름에 적합한 제어 구조 설정, 전반적인 흐름에 대한 오류 확인, 순서도의 최적화 과정이다. 제어 구조 설정 과정에서 조건 범위 및 포함 관계를 도식으로 정확히 설정한다. 흐름에 대한 오류 발견과 순서도 최적화 과정에서는 기술된 순서도의 문제 해결에 대한 흐름 및 과정의 오류를 발견하고 연산자를 사용하거나 불필요한 명령어 사용 여부를 파악한다.

개발단계는 순서도를 이용하여 파이썬 문법으로 변경하고 실행하는 파이썬 작성 및 실행 단계이다. 개발단계는 컴퓨팅 요소 중 자동화에 해당하는 부분으로 실제 고급 프로그래밍 언어로 작성하고 실행하는 단계를 의미한다. 파이썬을 작성하는 과정에서는 변수를 선언하고 사용하며 제어 구조에 적합한 산술, 관계, 논리 연산자를 선택하고 활용할 수 있도록 한다. 그리고 입력, 출력, 제어 구조에 따른 각 문법을 활용하여 작성한다. 파이썬으로 변환하는 과정에서 파이썬의 심화 문법을 사용하지 않고 순서도의 각 도형에 적합한 파이썬 문법을 간략히 사용하여 변환할 수 있도록 유도한다. 파이썬에 대한 심화 문법은 각 주차에 해당하는 심화 문법을 활용하여 적용 및 수정하는 시간을 갖기 위해서이다. 또한 작성한 파이썬을 실행함으로써 의도한 결과를 확인하고 오류 발생 시 오류 해결을 통하여 문법 및 흐름에 대한 오류를 이해하는 시간을 갖는다.

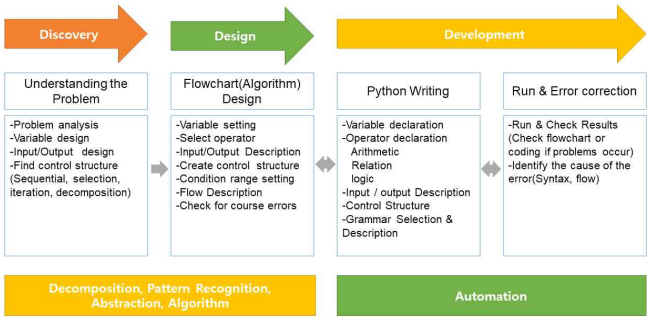

그림 2. 구조적 소프트웨어 교육 학습 모형의 진행 과정
Fig. 2. Progress of Structured Software Education Learning Model

표 10. 구조적 소프트웨어 교육 모형의 학습 단계모형
Table 10. Learning Step Model of Structured Software Education Model

Week	Step	Contents
1~2 Week	Pre-Learning	-Learning flowchart -Understand the flow chart, types and how to use
		-Learn Python Basic Grammar
3~7 Week	Lecture and Example Practice	-Weekly Python Deepening Learning -Understanding the problem -Write a flowchart -Python conversion -Run and fix errors
	Practice problem	-Problem analysis and understanding -Write a flowchart -Python conversion -Run and fix errors
	Feedback	-Find Flowchart Misconceptions -Modify Flowchart Flow -Contextual modification using Python deep syntax

구조적 소프트웨어 교육 학습 모형의 전반적인 진행 과정은 그림 2 와 같다. 제안하는 학습 모형은 매시간 주어진 문제를 해결하는데 사용하는 방법으로 문제를 이해하고 분석된 자료를 바탕으로 순서도를 작성하며 파이썬으로 변환 및 실행하는 과정을 반복적으로 학습하는 모델이다. 이 학습 모형에 대한 학습 단계 모형과 7주 기반의 세부 주차별 내용은 표 10 와 표 11과 같다.

매주 차시 학습 모형을 적용하기 위하여 선 학습으로써 1~2주차에는 알고리즘의 표현 방법, 순서도 이해 기호 및 작성 방법, 파이썬의 기본 문법을 먼저 학습한다. 여기서 파이썬의 각 문법은 짧은 시간에 쉽게 배울 수 있도록 핵심 문법을 작은 규모로 한번에 학습한다. 예를 들어 파이썬의 반복문이 While문과 for문이 있다면 선 학습에서는 for문만을 학습할 수도 있다. 그 이유는 첫째, 전반적인 문법 구조를 통하여 순서도를 파이썬 문법으로 변환하는 능력을 키우고 둘째, 주차가 진행되면서 각 문법에 대한 심화 과정을 통하여 구체적으로 문법에 대한 사용법을 익히고자 하는 데 있다. 셋째, 전반적인 문법을 간략히 반복함으로써 각 문법의 내용을 쉽게 익히도록 하는데 있다. 선 학습 단계에서는 간략하고 단순한 예제의 순서도를 파이썬 문법으로 변화하는 준비단계이다. 따라서 실습 예제의 경우 주어진 문제에 대한 문제 이해, 순서도 작성, 파이썬 작성에 대한 전반적인 단계를 수행하거나 순서도 빈 부분 채우기, 결과 예상하기 등과 같은 방법으로 문제해결과정을 기술할 수 있는 시간을 갖을 수도 있다.

표 11. 비전공자를 위한 구조적 소프트웨어 교육 모형의 세부 주차별 내용
Table 11. Detailed weekly content of structured software education model for non-majors

week	Theory lecture	Design & Programming
1	-Understanding Computing Thinking -Components of Computing Thinking	-Algorithm Representation Method -Understanding the Flowchart -A quick look at Python's basic syntax
2	Understanding Software & Coding	-Notes on Creating Flowcharts and Python -Conversion from flowchart to Python -Practice problem
3	Problem Analysis and Expression	-input/output -Variable and List Deepening Process -Conditional sentence deepening course -Practice problem
4	Decomposition of the Problem	-Recurring Course -Practice problem
5	Pattern Recognition	-Conditional statements and loops -Practice problem
6	Abstraction	-Function deepening process -Practice problem
7	Algorithm	-Practice problem -Final Test

선 학습된 자료를 바탕으로 3~7주차시에는 파이썬 문법을 분리하여 단계적으로 심화 학습을 진행한다. 매주 차시 학습 예제를 통하여 문제 이해, 순서도 작성, 심화 문법을 이용한 파이썬 작성 단계의 예를 통하여 작성 단계를 학습하고 실전 문제를 학습자에게 제시하여 스스로 문제를 해결할 수 있는 시간을 제공한다. 연습문제를 통한 결과에 대해서는 개별 또는 전체에 대한 피드백을 진행한다. 이 과정에서는 순서도 및 파이썬을 작성하는 단계에서 학습자가 갖는 잘못된 개념에 대한 순서도의 오류 이해와 파이썬 프로그램의 최적화 방법을 통하여 고급 프로그래머들의 프로그램 작성 방법들을 알려줄 수 있는 계기를 마련함으로써 프로그램의 흥미와 성취를 느끼도록 하기 위한 것이다[20].

V. 결론 및 향후 연구

4차산업에서 기술 간의 융합은 정보통신의 발달과 더불어 소프트웨어의 확산에 있다. 디지털 시대를 살아가는 학습자들에게 필수 학습요소로 자리매김하고 있는 소프트웨어 교육은

학습자에게 절차적이며 구조적인 새로운 문제해결 접근 방법을 알려줄 수 있을 뿐만 아니라 교육 방법에 따라 산업간의 융합을 위해 타인과 협업하여 최적의 해결안을 모색할 수 있는 의사소통 능력도 배양할 수 있다. 하지만 프로그래밍 언어 학습을 통한 소프트웨어 교육 중 텍스트 기반의 고급 프로그래밍 학습은 학습자에게 개발 환경의 부족, 문법 이해의 어려움, 컴퓨팅 사고력과 프로그래밍의 관계 부재등을 이유로 학습 저하로 이어지고 있다.

본 연구에서는 파이썬 기반의 프로그래밍 수업을 통하여 비전공자 학습자의 특성을 파악하고 이를 극복하기 위한 구조적 소프트웨어 학습 모형을 제시하였다. 제안한 학습 모형은 컴퓨팅적 사고에 대한 이해, 다이어그램을 통한 알고리즘 기술, 다이어그램을 기반으로 파이썬 프로그래밍으로의 변환, 피드백의 순서로 매주 학습이 이루어지도록 하였다. 학습자가 컴퓨팅 사고력을 이용하여 문제를 분석하고 문제 해결 흐름을 도형으로 표현한 후 파이썬 프로그래밍 문법으로 단계적으로 옮김으로써 파이썬에 대한 문법 이해의 아려움을 극복해보고자 하였다. 뿐만 아니라 피드백을 통하여 고급 프로그래머들의 소프트웨어 설계 원칙을 학습함으로써 좀 더 체계적이고 구체화된 프로그램 작성이 가능하도록 설계하였다. 향후 제안한 학습 모형을 통하여 연구결과를 도출하고 분석하여 보편적인 소프트웨어 교육을 위한 교수학습 설계를 진행하고자 한다. 또한, 프로젝트 기반의 교수학습 설계를 추가 적용하여 효과성을 분석하고자 한다.

참고문헌

[1] SW Education Plan in the 4th Industrial Revolution, National Information Society Agency, Intelligent Research Series, 2017.

[2] J. M. Wing, “Computational Thinking,” *Communications of the ACM*, Vol. 49, No. 3, pp. 33-35, 2010.

[3] J. S. Sung, S. H. Kim, H. C. Kim, “Analysis of Art and Humanity Major Learners Features in Programming Class,” *The Journal of Korean association of computer education*, Vol. 18, No. 3, pp. 25-35, 2015.

[4] S. H. Kim, “Analysis of Non-Computer Majors’ Difficulties in Computational Thinking Education,” *The Journal of Korean association of computer education*, Vol. 18, No. 3, pp. 49-57, 2015.

[5] J. W. Choi, Y. J. Lee, “The analysis of Learners` difficulties in programming Learning,” *The Journal of Korean association of computer education*, Vol. 17, No. 5, pp. 89-98, 2014.

[7] K. S. Oh, A study on the contents of computational thinking for programming education, Ph.D. Sungkyunkwan University, 2016.

[8] K. S. Oh, S. J. Ahn, “A study on the relationship between difficulty in learning to program and Computational Thinking,” *The Journal of Korean association of computer education*, Vol. 18, No. 5, pp. 55-62, 2015.

[9] KERIS, Training Textbooks for Software Leading Teachers, Elementary Schools KERIS, http://lib.keris.or.kr/search/detail/CAT000000013040.

[10] CSTA & ISTE, Computational thjnking in K-12 education teacher resource second edition, CSTA & ISTE, https://www.iste.org/docs/ct-documents/ct-teacher-resources_2ed-pdf.pdf?sfvrsn=2.

[11] J. K. Kim, “Thinking and programming as a college culture,” *Korea Multimedia Society Journal*, Vol. 22, No. 1, pp. 20-25, March 2018.

[12] W. S. Kim, “A Study on the Students Perceptions Trend for Software Essentials Subject in University,” *Korean Journal of General Education*, Vol. 13, No. 4, pp.161-180, August 2019.

[13] M. J. Oh, “Non-Major Students’ Perceptions of Programming Education Using the Scratch Programming Language,” *The Journal of Korean association of computer educatio*n, Vol. 20, No. 1. pp. 1-11, 2017.

[14] S. Y. Pi, “ A Study on Coding Education of Non-Computer Majors for IT Convergence Education,” *Journal of Digital Convergence*, Vol 14, No. 10, pp. 1-8, October 2016.

[15] K. S. You, S.M, Kim, K. C. Kim, S. Y. Choi, “ The Analysis of Learning Demotivation according to Gender and Programming Subjects in Programming Class’ Students of Liberal Arts,” *Journal of the Korea Institute of Information and Communication Engineering*, Vol. 23, No. 6, pp.704-710, June 2019.

[16] M. J. Lee, “Exploring the Effect of SW Programming Curriculum and Content Development Model for Non-majors College Students : focusing on Visual Representation of SW Solutions,” *Journal of Digital contents Society*, Vol. 18, No. 7, pp. 1313-1321, November 2017.

[17] Y. S, “Python-based Software Education Model for Non-Computer Majors,”*Journal of the Korea Convergence Society*, Vol. 9, No. 3, pp. 73-78, 2018.

[18] http://www.python.org

[19] W. S. Kim, “Exploring the direction of glandular basic-software education considering the major of college students,” J*ournal of The Korean Association of Information Education*. Vol. 23, No. 4, pp. 329-341, August 2019.

[20] Y. M. Choi, “Analysis and Application of Misconception Flowchart for Programming Control Structure Concept Learning,” J*ournal of Korea Multimedia Society*, Vol. 20, No.12, pp. 2000-2008, 2017.

[21] M. Piteira, C. J. Costa, “Learning computer programming: study of difficulties in learning programming,” *in Proceedings of the 2013 International Conference on Information Systems and Design of Communication,* Lisboa, Portugal, pp.75-80, 2013.

[22] S. j. Ahn, K. S. Noh, K.S. Oh, *Computational Thinking*, EhanMeida, 2019.

[23] W. S. Moon, “Analysis of error data generated by prospective teachers in programming learning, *Journal of the Korean Association of Information Education*, Vol. 22, No. 2, pp. 205-212, 2018.

[24] S. P. Davies, “Models and theories of programming strategy,” *International Journal of Man-Machine Studies,* Vol. 39, No. 2, pp. 237-267, 1993.

[25] H. J. Choe, “The Programming Education Framework for Programming Course in University,” *The Journal of Korean association of computer education,* Vol. 14, No. 1, pp. 69-79, 2011.

[26] E. S Jung, M. Huh, H. J. Young, Y. S. Kim, “Effect of a Flow Char Learning on Logical Thinking Ability and Performance Achievement in Middle School Computer Programming Class,” *The Journal of Korean association of computer education,* Vol. 1212, No. 6, pp. 11-19, 2009.

[27] Y. M. Kim, M. J. Lee, “A Comparative Study of Educational Programming Languages for Non-majors Students: from the Viewpoint of Programming Language Design Principles ,” *The Journal of Korean association of computer education,* Vol. 22, No. 1. pp. 47-61, 2019.

[28] J. Kim et al,. (2016). 2015 Education Policy Network Training on-site support research : Development of SW Education Teaching and Learning Model. Commissioned research CR 2015-35.

강의선(Eui-Sun Kang)

2002년 : 숭실대학교 (공학석사)
2007년 : 숭실대학교 (공학박사-미디어공학)

2007년～현 재: 숭실대학교 베어드교양대학 부교수
※관심분야 : 멀티미디어(Multimedia), 사물 인터넷(Internet of Things), 컴퓨터 교육(Computer Education) 등