https://doi.org/10.32431/kace.2019.22.6.005

컴퓨팅 사고력 기반 테스트 중심 문제해결 학습 모형 연구

김영직† · 김성식††

요 약

4차 산업혁명으로 초지능, 초연결 사회가 도래했다. 4차 산업혁명의 핵심 기술들은 소프트웨어가 중요한 부분을 차지한다. 소프트웨어를 통해 문제를 해결하는 능력은 모든 사람이 갖추어야 할 핵심 역량이란 점에서 소프트웨어 교육이 필요하다. 소프트웨어 교육은 프로그래밍 과정을 통해 문제를 해결하는 능력인 컴퓨팅 사고력 향상을 목표로 한다. 지금까지 대부분의 문제해결 프로그래밍 교수학습 모형은 전통적 개발 방식인 폭포수 모형(분석-설계-개발-테스트)을 따른다. 이는 선행 단계에서 문제가 있는 경우 테스트 단계에서 발견되어 문제의 해결책을 다시 찾는 데 적지 않은 시간과 노력이 소요되어 학습에 부담으로 작용할 수 있다. 본 연구에서는 애자일(Agile) 개발 방식인 TDD(테스트주도개발)를 적용한 컴퓨팅 사고력 기반 테스트 중심 문제해결 학습 모형을 제안하고 전문가 검토를 통해 모형의 적절성을 검증하였다. 모형의 검증 결과 긍정적인 평가 결과를 보였다. 특히, 모형의 학습 단계 구성, 프로그래밍 학습에의 도움, 컴퓨팅 사고력 증진에의 도움 등에서는 높은 평점을 보여 향후 학습 적용 시 문제해결 프로그래밍 학습을 통한 컴퓨팅 사고력 발달에 긍정적인 효과가 있을 것으로 판단된다.

주제어 : 컴퓨팅 사고력, 테스트 중심 문제해결 학습 모형, CT-TDPS, 테스트 주도 개발

A Study on Computational Thinking based Test-Driven Problem Solving Learning Model

Young-Jik Kim† · Seong-Sik Kim††

ABSTRACT

In the Fourth Industrial Revolution, a super-intelligent and super-connected society has arrived. Software is an important part of the core technologies of the Fourth Industrial Revolution. The ability to solve problems through software requires software education in that it is a core competency that everyone should have. Software education aims to improve Computational Thinking, which is the ability to solve problems through programming. Until now, most problem-solving programming learning models follow the traditional method of development: Waterfall model (Analysis-Design-Development-Test). In this model, if there is a problem in the preceding step, That could be found in the test phase. This takes a considerable amount of time and effort to find a solution to the problem and can be a burden on the programming learning. In this study, we proposed a Test-Driven Problem-Solving learning model using TDD (Test Driven Development) as Agile development method, and reviewed the appropriateness of the model through experts review. The verification results of the model showed positive evaluation results. In particular, the learning phase configuration of the model, helping in programming learning, helping of Computational Thinking improvement showed high rating, it is determined that there will be positive effects on Computational Thinking improvement through problem-solving programming learning when applying future learning.

Keywords : Computational Thinking, Test Driven Problem Solving Learning Model, CT-TDPS, Test Driven Development

†정 회 원: 한국교원대학교 컴퓨터교육과 박사과정
††종신회원: 한국교원대학교 컴퓨터교육과 교수(교신저자)
논문접수: 2019년 10월 4일, 심사완료: 2019년 10월 31일, 게재확정: 2019년 11월 12일

1. 서론

컴퓨터와 인터넷의 등장으로 우리는 지식정보사회를 살아가고 있다. 이제는 사물이 초지능을 갖고, 서로 연결되는 초연결 사회가 눈앞에 펼쳐지고 있다. 4차 산업혁명으로 촉발된 이러한 변화는 과거 산업혁명과는 다르게 컴퓨터를 중심으로 빠르게 진행되고 있다.

국가적으로도 4차 산업혁명의 핵심 기술에 적극적으로 투자하고 있다. 블록 체인, 드론, 3D 프린팅, 자율주행자동차, 인공지능, 로봇, 스마트팩토리 등은 최근 기술 발전이 놀랍다. 이러한 핵심 기술에는 소프트웨어가 중요한 부분을 차지한다.

소프트웨어는 인간이 사고를 모델링하고 추상화하고 자동화하는 강력한 도구이다. 소프트웨어가 산업, 경제, 학문, 교육 등 거의 대부분의 영역에서 경쟁력을 결정하는 소프트웨어 중심 사회에서 소프트웨어를 활용하고, 개발하는 방법을 배우는 소프트웨어 교육은 선택이 아니라 필수이다[1].

소프트웨어를 통해 문제를 해결하는 능력은 모든 사람이 갖추어야 할 핵심 역량이란 점에서 소프트웨어 리터러시로 인식되고 있다[2]. 소프트웨어 교육이 필요한 이유가 바로 이 때문이다.

전 세계적으로 소프트웨어 교육이 확산되고 있다. 우리나라에서도 2015 개정 교육과정을 통해서 초중등에서 소프트웨어 교육을 강화하였다. 개정 교육과정에서의 소프트웨어 교육은 소프트웨어의 사용법이나 프로그래밍 언어의 문법 학습은 최소화하고, 문제해결 프로그래밍을 통한 컴퓨팅 사고력 신장에 초점을 맞추고 있다[3].

소프트웨어 교육에서 강조하는 것이 바로 컴퓨팅 사고력이다. 컴퓨팅 사고력은 인간 관점이 아닌 컴퓨터 관점에서 문제를 해결한다는 것이 특징이다. 컴퓨팅 사고력은 문제 해결책을 컴퓨터에게 알려주어 컴퓨터로 하여금 문제를 정확하고 빠르게 해결하도록 하는 행위 과정 전반에서 발달한다.

문제해결 프로그래밍 학습을 통해서 컴퓨팅 사고력이 발달하기에 다양한 컴퓨팅 사고력 기반 문제해결 교수학습 모형이 등장하였다.

지금까지의 컴퓨팅 사고력 기반 교수 학습 모형은 분석-설계-구현-테스트(디버깅)라는 전형적인 소프트웨어 개발 라이프 사이클(SDLC)인 폭포수 모형의 프로그래밍 개발 절차를 따르고 있다.

폭포수 모형에서는 테스트가 마지막 단계에서 이뤄진다. 이는 선행 단계에서 문제가 있거나 변경이 발생했을 때 이를 해결하는데 시간과 노력이 많이 소요된다는 단점이 있다. 최근에는 개발 주기를 빠르게 반복하면서 반복적·점증적으로 개발하는 방식을 선호한다[4][13].

본 연구에서는 최근 프로그래밍 개발에서 활용도가 높은 애자일(Agile) 개발 방식에서 사용되는 TDD(Test Driven Development)를 적용한 테스트 중심의 문제해결 프로그래밍 학습 모형을 제안하였다.

2. 이론적 배경

2.1 컴퓨팅 사고력

컴퓨팅 사고력(Computational Thinking)은 Seymour Papert(1996)가 기하학적 아이디어 생성을 위한 접근방법으로 처음 사용하면서 우리에게 소개되었고 그 후 Wing에 의해 알려지게 되었다[5].

Wing(2006,2008)은 그의 연구에서 추상화 및 자동화 등의 컴퓨팅을 활용한 사고능력 및 문제해결의 과정을 Computational Thinking이라고 제안하였고, 많은 연구자들은 이를 컴퓨팅을 통한 새로운 문제해결 방법론이라고 주장하며 이를 다양한 교과교육에 적용하기 위한 연구를 진행하고 있다[6].

컴퓨팅 사고는 컴퓨터 과학의 원리와 개념을 바탕으로 문제를 해결하는 연속적이고 체계적인 사고의 과정이다. 컴퓨팅 사고의 요소들은 컴퓨터 과학적 문제해결과정에서 유기적이고 순환적으로 작동한다[1].

이철현(2016)은 대표적인 컴퓨팅 사고력의 구성요소를 <표 1>과 같이 정리하였다.

<표 1> 컴퓨팅 사고력 구성요소

연구자(년도)	컴퓨팅사고력 구성 요소
Wing(2006)	• 문제표현 • **문제 재정의** • **재귀적 사고** • **분해** • **추상화** • **발견적 추론 생성** • **모듈화** • 선행적 사고 • **시간과 공간의 효율화**

ISTE & CSTA (2011)	• 자료 수집 • 자료 분석 • 자료 표현 • **문제 분해** • **추상화** • **알고리즘과 절차** • **자동화** • 시뮬레이션 • 병렬화
Brennan & Resnick (2012)	• **개념** : 시퀀스, 반복, 병렬처리, 이벤트, 조건, 연산, 데이터 • **실습 : 증진과 반복, 테스팅과 디버깅, 재사용과 재조합, 추상화와 모듈화** • **관점** : 표현하기, 연결하기, 질문하기
영국 Key State 3 (2016)	• **분해** • **패턴인식** • **추상화** • **알고리즘**
Google (2016)	• **분해** • **패턴인식** • **추상화** • **알고리즘 설계**
교육부·KERIS (2015)	• 자료 수집 • 자료 분석 • **구조화** • **추상화** : 분해, 모델링, 알고리즘 • **자동화** : 코딩, 시뮬레이션 • **일반화**

컴퓨팅 사고력의 하위 구성 요소들은 추상화와 자동화로 분류하고 그 하위에 여러 요소들을 대응할 수 있다. 컴퓨팅 사고력의 각 내용과 요소들은 학습의 상황에 따라 다양하고 능동적으로 적용되고 있다[7].

2.2 문제해결 프로그래밍 교수 학습 모형

문제해결 프로그래밍은 기존의 단편적인 개념을 확인하기 위해 프로그래밍하는 것과는 차별화된다. 문제 상황제시, 입력과 출력 방법, 입출력 사례 제시 등 주어진 문제를 보다 효율적이며 일반적으로 해결하기 위해 프로그래밍 문법, 자료구조, 알고리즘 등을 사용하여 프로그래밍하는 것으로 정의할 수 있다[8].

문제 중심 스크래치 프로그래밍 수업 모형의 학습 절차는 5단계로 구성되며 <표 2>와 같다[9].

국·내외 컴퓨팅 사고력 기반의 프로그래밍 교육 교수 학습 모형의 학습 절차를 정리하면 <표 3>과 같다[7].

<표 2> 문제 중심 스크래치 프로그래밍 수업 모형의 학습 절차

학습 절차	활동 내용
문제 인식하기	• 동기 유발 • 문제 제시 및 파악 • 기본 프로그래밍 자료 소개
문제해결책 계획세우기	• 현재 알고 있는 내용 정리 • 문제를 해결하기 위해 알아야 할 내용 파악 • 문제 해결을 위한 일정, 역할 분담 계획
탐색하기	• 문제 해결을 위한 관련 지식, 정보 탐색 • 문제해결계획의 검토 • 문제해결계획의 수정
해결책 발견하기	• 다양한 문제 해결책 고안하기 • 최적의 프로그래밍 결과물(문제 해결책) 결정하기
발표 및 평가하기	• 해결책을 모둠별로 발표 • 자기평가와 상호평가, 과정평가와 결과평가 • 문제해결과정에 대한 자신의 참여도 및 태도에 대한 평가

<표 3> CT 기반 교수학습 모형의 학습 절차

연구자	교수학습 모형	학습 절차
KERIS (2015)	CT요소중심모델 (DPAAP)	• **분해** • **패턴인식** • 추상화 • **알고리즘** • **프로그래밍**
최숙영 (2016)	컴퓨팅사고력 기반 수업 프레임워크	• **문제인식** • **문제정의** • **해결책고안** • **문제해결의 실천** • **문제해결의 평가**
이철현 (2016)	CT-PS	• **문제 분석** • 추상화 • **구현**
전용주 (2017)	CT-CPS	• **문제 인식 및 분석** • **아이디어 구상** • **설계** • **구현 및 적용**
Tauno Palts (2015)	컴퓨팅 사고력 학습모델	• 요구 또는 문제 식별 • 요구 또는 문제 연구 • 솔루션 개발 • 최적의 솔루션 선택 • 프로토타입 제작 • **테스트 & 솔루션 평가** • **솔루션 상호소통** • 재설계

선행 연구를 통해 문제해결 프로그래밍의 학습 절차를 정리하면 <표 4>와 같다.

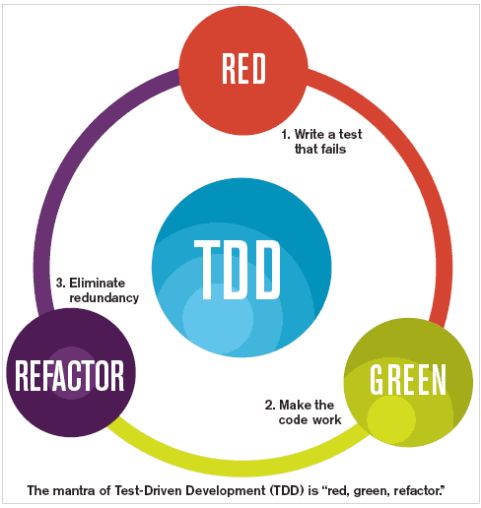
<표 4> 문제해결 프로그래밍 학습 절차

학습 절차	활동 내용
문제 분석	• 문제 인식 및 분석 • 문제 분해
문제 정의	• 문제 정의
해결책 고안, 아이디어 구상	• 탐색하기 • 패턴인식 • 해결책 고안 • 아이디어 구상 • 설계
해결책 실천, 구현	• 알고리즘 • 문제 해결책 실천 • 구현 및 적용 • 프로그래밍
평가	• 테스트 & 솔루션 평가 • 솔루션 상호소통 • 공유

2.3 테스트주도개발 (TDD:Test Driven Development)

XP(eXtreme Programming) 창시자 중 한명이며, TDD를 주도한 Kent Beck은 자신의 저서에서 TDD는 '프로그램을 작성하기 전에 테스트를 먼저 작성하는 것'이라고 테스트 주도 개발을 정의했다[14]. TDD는 최근 XP에서 많이 사용되고 있는 코드 개발 전략이다[15].

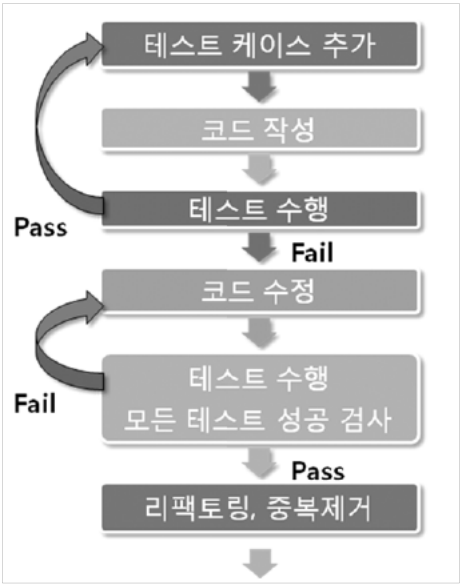
테스트주도개발의 진행 방식은 크게 3단계로 나뉘는데 각 단계의 결과 상태에 따라 '실패-성공-리팩토링'으로 표현된다.



[그림 1] TDD 개발 방식[16]
(Red:실패, Green:성공, Refactor:리팩토링)

전통적인 개발 방식인 폭포수 모형은 요구사항분석-설계-구현-테스트-유지보수 과정을 거치는데 반해 TDD는 처음부터 테스트 코드를 작성하는 것이 특징이다[10].

프로그램의 품질을 향상시키기 위해서 초기부터 테스트 계획을 미리 작성한다. 미리 작성된 프로그램 테스트 케이스는 짧은 반복 개발 주기를 통해 테스트 실패 또는 성공 과정을 반복하게 된다. 테스트 케이스에 만족하는 코딩이 작성되면 개발자는 리팩토링을 통해 프로그램을 최적화 시키는 과정을 거친다[11].



[그림 2] 테스트주도개발 절차 [11]

TDD의 장점으로 첫째, 개발의 방향을 잃지 않게 유지해준다. 자신이 어떤 기능을 개발하고 있고 어디까지 개발했는지를 항상 살펴볼 수 있다. 남은 단계와 목표를 잊지 않게 도와준다. 둘째, TDD를 통해 만들어진 프로그램은 품질이 우수하다. 이는 필요한 만큼 테스트를 거친 '품질이 검증된 부품'을 갖게 되는 것과 마찬가지다[12].

TDD를 프로그래밍 교육에 적용한 연구는 국내에서는 거의 찾아볼 수 없다. 하지만 국외에서는 컴퓨터 과학(CS) 교육과정에 TDD를 적용한 사례를 쉽게 찾아 볼 수 있다.

TDD는 작은 프로젝트에도 적용가능하고 대학생 대상의 프로그래밍 교육에서 코딩에 대한 자신감을

크게 향상시켰다. 파스칼 프로그래밍 교육에서 TDD를 적용한 집단이 그렇지 않은 집단보다 코드의 결함률이 낮고, 완성도가 높았다[14].

TDD는 산업분야에서 사용되는 소프트웨어 개발 방식으로 컴퓨팅 교육 분야에서도 사용되고 있다. 대학에서 CS1 기초, CS2 과목에서 TDD를 가르친다. 적절한 교육적 시스템 기반에서 학생들이 TDD를 활용했을 때 코드와 테스팅의 질을 향상시킬 수 있다[15].

자동화 단위 테스트를 활용한 예제 중심 컴퓨팅 프로그래밍 교육에서 TDD방식이 전통방식(Test-Last) 보다 소프트웨어의 품질이 향상되었다. 컴퓨터과학 교육과정에 TDD를 반영하여 학생들의 프로그램 설계 능력과 테스팅 능력이 향상될 수 있도록 TDD에 대한 폭넓은 교육적 접근이 필요하다[17].

김영직(2017)은 TDD를 활용한 프로그래밍 교육 고찰 연구에서 문제중심학습(PBL) 모형에 기반한 TDD 프로그래밍 학습 모형 개발의 필요성을 제안하였다[10].

선행 연구를 통해서 문제해결 프로그래밍 학습 절차와 테스트주도개발 절차를 고려한 학습 단계를 정리하면 <표 5>와 같다.

<표 5> 문제해결 프로그래밍 학습 절차와 테스트주도개발 절차를 고려한 학습 단계

학습 단계	문제해결 프로그래밍 학습 절차	테스트주도개발(TDD) 절차
문제 정의 및 추론	• 문제 분석	
	• 문제 정의	
	• 해결책 고안 • 아이디어 구상	
코드 구현	• 해결책 실천 • 구현	• Write a test that fails (실패) • Make the code work (성공)
코드 개선	• 평가	• Eliminate redundancy (리팩토링)

3. 컴퓨팅 사고력 기반 테스트 중심 문제해결 학습 모형

3.1 학습 모형 개발 전략

학생은 문제를 해결하기 위하여 아이디어를 구상하고, 해결 방법을 찾아 알고리즘을 만들고, 이를 프로그래밍하는 과정 전반에 어려움을 느낀다. 각 과정에서 해야 할 목표를 명확히 하고 반복적으로 수행하는 과정에서 즉각적인 피드백을 받을 수 있도록 하여 학습에 집중할 수 있도록 한다.

교사는 학생이 만든 프로그램을 평가할 수 있는 지표가 필요하다. 프로그래밍 과정과 프로그램 실행 결과에 대한 객관적인 평가 지표를 제공하도록 한다.

해결해야 하는 문제는 프로그래밍의 개념과 원리를 이해하는 것에서부터 실생활에서 해결해야 하는 문제까지 다양하게 구성할 수 있다.

학생과 교사의 관점을 고려하여 모형의 개발 전략은 다음과 같다.

① 학생 중심의 자기주도적인 학습이 가능하도록 컴퓨팅 사고력 기반의 문제해결 프로그래밍 학습 절차를 기반으로 한다.

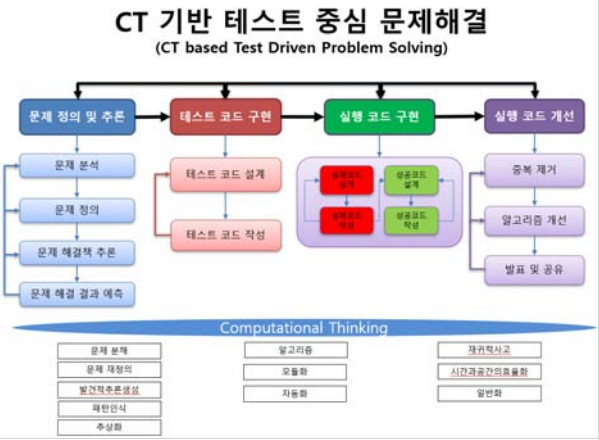
② 문제를 세분화하고 목표를 명확히 정의한 후 점증적으로 프로그래밍하면서 실행 결과에 대한 즉각적 피드백(실패, 성공)을 확인할 수 있는 테스트주도 프로그래밍 방법을 구현 절차에 반영한다.

③ 학습 단계마다 중간 산출물을 제시하고 이를 프로그래밍 과정의 평가 지표로 활용할 수 있도록 하며, 프로그램의 완성도와 문제해결 평가는 테스트 코드를 활용할 수 있다.

3.2 학습 모형 개발

선행 연구를 통해 문제해결 프로그래밍 학습 절차와 테스트주도개발 절차를 고려한 학습 단계를 적용하여 4단계로 구성하였다.

문제 정의 및 추론, 테스트 코드 구현, 실행 코드 구현, 실행 코드 개선이다.



[그림 3] 컴퓨팅 사고력 기반 테스트 중심 문제해결 학습 모형

3.2.1 1단계 : 문제 정의 및 추론

문제를 분석하고 분해하여 하위 단위 문제들로 정의한다. 정의된 하위 단위 문제의 해결 과정을 추론하여 해결책을 구상하고 입력값에 따른 결과값을 예측한다.

1) 문제 분석

문제를 분석하여 하위 단위 문제를 식별한다.

2) 문제 정의

하위 단위 문제들을 정의한다.

3) 문제 해결책 추론

나열하기, 세어보기, 관찰 등을 통해 문제 해결의 핵심 원리를 발견하고 규칙성과 유사성을 찾아 해결책을 추론한다.

4) 문제 해결 결과 예측

하위 단위 문제별로 입력값에 따른 결과값을 예측한다.

3.2.2 2단계 : 테스트 코드 구현

1단계에서 정의된 하위 단위 문제별로 테스트 코드를 설계한다. 테스트 코드는 실행 코드의 안정성과 정확성을 미리 확보하기 위한 목적을 갖으며 프로그램의 평가 지표로도 활용 가능하다.

1) 테스트 코드 설계

1단계에서 정의된 하위 단위 문제별로 테스트 방법을 구상하고, 모듈을 설계한다.

2) 테스트 코드 작성

설계를 바탕으로 테스트 모듈을 작성한다. 테스트 모듈은 3단계에서 개발할 실행 코드와 연계하여 테스트 결과가 올바른지 확인할 수 있다.

3.2.3 3단계 : 실행 코드 구현

2단계에서 구현한 테스트 모듈을 만족하는 실행 모듈을 설계하고 코드를 작성한다. 하위 단위 문제별로 하나씩 작성하면서 점증적으로 확장할 수 있다. 실행 코드는 테스트 코드를 최대한 빠르게 성공 시키도록 작성한 후, 구상한 해결책(알고리즘)을 설계에 반영하여 실행 코드를 작성한다.

1) 실패 코드 설계

실행 모듈을 설계한다(모듈 이름, 입력 변수 선언).

2) 실패 코드 작성

실행 모듈을 작성하고 결과 변수의 값을 '0'으로 설정한다. 이 때 테스트 결과는 'FAIL'이다.

3) 성공 코드 설계

실행 모듈을 빠르게 성공시키도록 결과 변수의 값을 희망값으로 지정할 수 있다. 해결책을 구상하고 알고리즘을 설계하여 의사코드를 작성한다.

4) 성공 코드 작성

희망값을 이용하여 실행 모듈을 빠르게 성공시킨 후에는 실제 알고리즘을 적용하여 코드를 작성한다. 이 때 중요한 것은 테스트 결과가 성공인지 확인하면서 코드를 작성한다. 이 때 테스트 결과는 'PASS'이다.

3.2.4 4단계 : 실행 코드 개선

실행 코드 구현을 통해서 모든 하위 단위 문제에 대한 테스트를 통과했다면 전체 실행 코드에서 중복된 코드를 찾아 제거하거나 알고리즘을 개선할 수 있다. 프로그램을 수정하는 동안 테스트 결과가 'PASS' 인지 확인하도록 한다. 개선할 필요가 없는 경우에는 중복제거와 알고리즘 개선을 생략할 수 있다.

최종 완성된 프로그램에 대해 개발 과정을 발표하고 공유하여 나의 해결책과 다른 해결책을 비교해 본다.

1) 중복 제거

실행 코드의 중복성을 제거한다. 해당 부분의 테스트 결과가 'PASS'인지를 확인하면서 진행한다.

2) 알고리즘 개선

수행 시간을 줄이고 메모리를 효율적으로 사용하도록 알고리즘을 개선한다. 해당 부분의 테스트 결과가 'PASS'인지를 확인하면서 진행한다.

3) 발표 및 공유

각자의 프로그램 개발 과정을 발표, 공유하고 서로의 해결책을 비교한다. 다른 해결책을 이용하여 알고리즘을 개선해 볼 수 있다.

<표 6> CT-TDPS 모형 학습 단계와 산출물

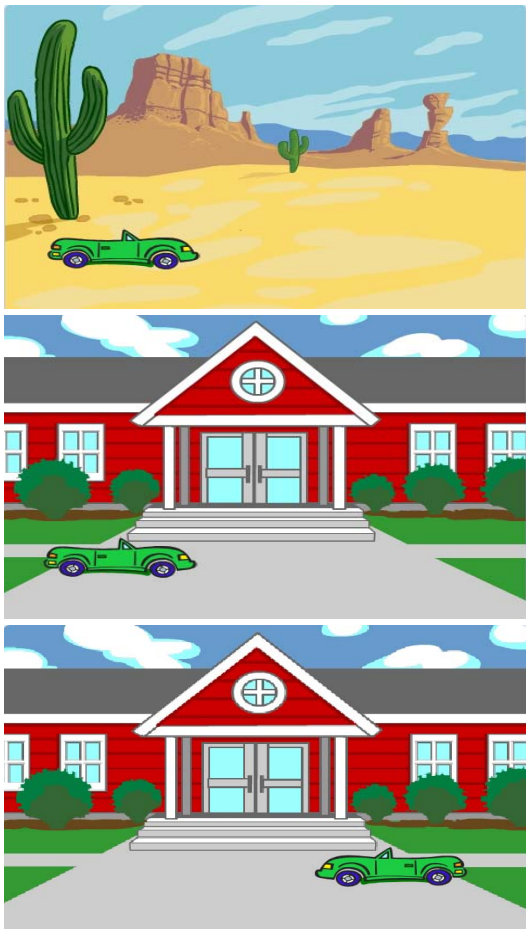
학습 단계	하위 단계	산출물
1. 문제 정의 및 추론	문제 분석	문제 정의서
	문제 정의	
	문제 해결책 추론	해결 아이디어 및 해결 절차
	문제 해결 결과 예측	입력처리출력표
2. 테스트 코드 구현	테스트 코드 설계	테스트 모듈 설계서
	테스트 코드 작성	테스트 코드
3. 실행 코드 구현	실패 코드 설계	모듈 설계서
	실패 코드 작성	실행 코드(실패)
	성공 코드 설계	알고리즘
	성공 코드 작성	실행 코드(성공)
4. 실행 코드 개선	중복 제거	중복 제거된 실행 코드
	알고리즘 개선	개선된 알고리즘, 실행 코드
	발표 및 공유	비교평가서

4. 컴퓨팅 사고력 기반 테스트 중심 문제해결 학습 모형 예시

4.1 문제 예시

<문제> 자율주행자동차 만들기

멋진 자동차가 있습니다. 위치는 화면 좌측 하단 부근입니다. 배경은 사막입니다. 자동차가 앞으로 움직입니다. 자동차가 화면 끝에 닿으면 배경은 학교로 바뀝니다. 자동차는 화면 좌측 하단 부근에서 다시 앞으로 움직이기 시작합니다. 화면 끝에 닿으면 자동차의 방향만 반대로 바뀌고(배경 변화없음) 다시 앞으로 움직입니다. 화면 끝에 닿으면 배경은 사막으로 바뀌고 화면 우측 하단에서 좌측 방향으로 움직이다가 화면 끝에 닿으면 정차합니다. 자동차가 사막과 학교를 계속해서 번갈아 가며 반복 주행하도록 코딩하세요. 배경이 학교인 경우 속도를 느리게 주행하도록 합니다. 사막에선 원래의 속도로 주행합니다.



[그림 4] 문제 예시

4.2 CT-TDPS 모형 활동지 작성 예시

4.2.1 문제 정의 및 추론 (1단계)

1) 문제 분석

- 자동차는 좌측에서 우측으로 움직이거나 우측에서 좌측으로 움직인다.
- 자동차가 화면 끝에 닿으면 화면이 바뀌거나 또는 자동차의 방향이 바뀐다.
- 배경이 학교인 경우 자동차의 움직이는 속도는 줄어든다.
- 배경이 사막인 경우 자동차의 움직이는 속도는 원래의 속도를 갖는다.

2) 문제 정의

- (주행하기)자동차는 이동 방향으로 주행한다.
- (배경바꾸기)자동차가 화면 끝(사막)에 닿으면 배경 화면이 학교로 바뀐다.
- (시작위치정하기) 자동차는 처음 시작위치로 이동한다.
- (방향바꾸기)자동차가 화면 끝(학교)에 닿으면 자동차의 방향이 반대로 바뀐다.
- (속도정하기)배경 화면이 학교인 경우 자동차의 속도는 감속된다. 사막인 경우 원속도로 바뀐다.

3) 문제 해결책 추론

☞ 순서대로 실행 과정을 나열해 봅니다.

- 사막 배경-> 좌측에서 우측으로 주행하기(주행 속도는 10) -> 화면 끝에 닿으면 -> 학교 배경 변경 -> 좌측에서 우측으로 주행하기(주행 속도는 5) -> 화면 끝에 닿으면 -> 자동차의 방향이 반대로 바뀜(방향 -90) -> 학교 배경 -> 우측에서 좌측으로 주행하기(주행 속도는 5) -> 화면 끝에 닿으면 -> 배경이 사막으로 변경 -> 우측에서 좌측으로 주행하기(주행 속도는 10) -> 화면 끝에 닿으면(방향 90) -> 사막 배경 -> 좌측에서 우측으로 주행하기(주행 속도는 10) ....(계속 반복)

☞ 정의된 하위 문제를 이용해서 나열해 봅니다.

- 시작 -> 배경바꾸기(사막) -> 방향바꾸기(우측방향) -> 시작위치정하기(좌측끝)-> 속도정하기(10) -> 주행하기 -> 화면 끝 -> 배경바꾸기(학교) -> 방향바꾸기(우측방향) -> 시작위치정하기(좌측끝) -> 속도정하기(5) -> 주행하기 -> 화면 끝 -> 배경바꾸기(학교) ->방향바꾸기(좌측방향) -> 시작위치정하기(우측끝) -> 속도정하기(5) -> 주행하기 -> 화면 끝 -> 배경바꾸기(사막) -> 방향바꾸기(좌측방향) -> 시작위치정하기(우측끝) -> 속도정하기(10) - 주행하기 -> 화면 끝 -> 처음부터 반복

4) 문제 해결 결과 예측 (입력처리출력표 작성)

변수 / 상태	횟수	배경	시작 위치	방향	속도	
시작	1	사막	좌측	우	10	
벽	2	학교	좌측	우	5	반복
벽	3	학교	우측	좌	5	
벽	4	사막	우측	좌	10	
벽	5	사막	좌측	우	10	
벽	6	학교	좌측	우	5	반복
벽	7	학교	우측	좌	5	
벽	8	사막	우측	좌	10	
벽	9	사막	좌측	우	10	
벽	10	학교	좌측	우	5	반복
벽	11	학교	우측	좌	5	
벽	12	사막	우측	좌	10	

4.2.2 테스트 코드 구현 (2단계)

1) 테스트 코드 설계

횟수가 10 일 때 배경이 School 이어야 테스트 배경결과가 PASS

테스트배경 결과값 예측	테스트배경 의사코드
테스트배경 (입력값 : 횟수)	테스트배경 (횟수)
테스트배경(1) : 사막	만약 횟수 = 10
테스트배경(2) : 학교	만약 배경 = School
테스트배경(3) : 학교	테스트배경결과는 PASS
테스트배경(4) : 사막	아니면
...	테스트배경결과는 FAIL
☞ **테스트배경(10) : 학교**	멈춤

2) 테스트 코드 작성

테스트 설계대로 코드를 구현합니다.

[그림 5] 테스트 코드(배경)

4.2.3 실행 코드 구현 (3단계)

1) 실패 코드 설계(테스트 실패시키기)

배경바꾸기 설계(실패)	배경바꾸기 의사코드(실패)
배경바꾸기 입력값 : 횟수 배경을 무조건 0으로 정한다. (실패시키기)	배경바꾸기 (횟수) 배경을 0 으로 정하기

2) 실패 코드 작성

[그림 6] 실패 코드

3) 성공 코드 설계(테스트 성공시키기)

실패한 코드를 성공하도록 해결책을 알고리즘으로 설계하고 의사코드로 작성합니다.

배경바꾸기 설계(성공)	배경바꾸기 의사코드(성공)
배경바꾸기 입력값 : 횟수 횟수를 4로 나눈 나머지가 0이면 사막 횟수를 4로 나눈 나머지가 1이면 사막 횟수를 4로 나눈 나머지가 2이면 학교 횟수를 4로 나눈 나머지가 3이면 학교	배경바꾸기 (횟수) 만약 횟수 / 4 = 0 또는 횟수 / 4= 1 배경 = Desert 아니면 배경 = School

4) 성공 코드 작성

의사코드를 이용하여 실패 코드를 성공하도록 변경합니다.

[그림 7] 성공 코드

☞ 횟수 값에 따라서 배경이 정상적으로 바뀌는지 확인해 봅니다.

[그림 8] 배경바꾸기, 테스트배경 실행 코드

[그림 9] 배경바꾸기, 테스트배경 실행 결과 화면

☞ 횟수값이 1초 간격으로 1씩 증가하면서 배경 변수값이 바뀌는 것을 확인할 수 있습니다. 즉, 자동차는 아직 움직이지 않지만 횟수값에 따라서 배경이 정상적으로 바뀌고 있음을 확인할 수 있습니다(입력처리출력표의 배경값과 맞는지 확인합니다).

☞ 다음은 방향에 대한 테스트 코드와 실행 코드를 작성해 봅니다.

[그림 10] 방향바꾸기, 테스트방향 실행 코드

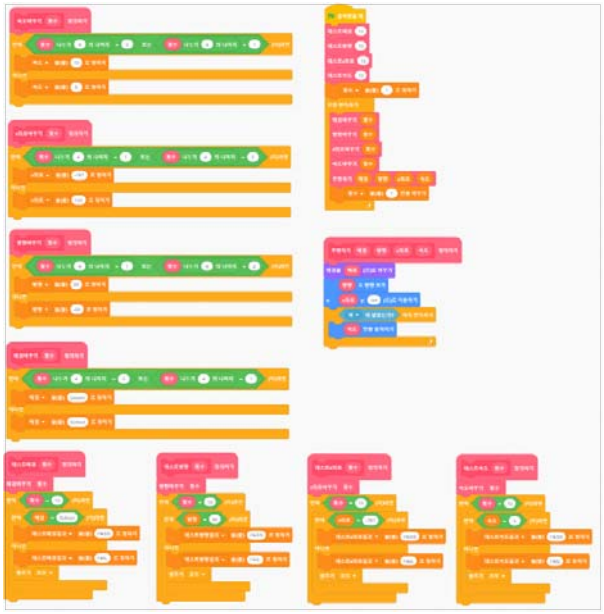
☞ 횟수가 1초 간격으로 1 씩 증가하면서 배경, 방향이 정상값으로 바뀌는 것을 확인할 수 있습니다.

[그림 11] 실행 결과 화면(배경, 방향)

☞ x좌표, 속도를 순서대로 테스트 코드와 실행 코드를 작성하여 추가합니다.

☞ 최종적으로 주행하기 모듈을 만들어서 실제 자동차를 주행시켜 봅니다.

[그림 12] 자율주행자동차 주행하기, 실행 블록



[그림 13] CT-TDPS 모형에 기반한 자율주행자동차 스크래치 블록(완성)

4.2.4 실행 코드 개선 (4단계)

1) 중복 제거

중복되어지는 코드가 있는지 살펴보고 중복을 제거하도록 프로그램을 변경합니다. 프로그램 변경이 될 때마다 테스트 결과가 성공인지 확인합니다.

2) 알고리즘 개선

중복 제거를 통해 각 함수의 코드를 개선할 수 있는지 살펴보고 변경하여 프로그램을 완성합니다. 프로그램 변경이 될 때마다 테스트 결과가 성공인지 확인합니다.

3) 발표 및 공유

모형을 통해 문제를 해결하는 과정에서 생각한 아이디어와 해결책을 이야기 합니다. 테스트 코드 구현, 실행 코드 구현, 개선 과정을 설명하고 최종 완성된 프로그램을 시연해 봅니다.

다른 사람의 발표를 들으면서 나의 해결책과 비교해 봅니다.

5. 컴퓨팅 사고력 기반 테스트 중심 문제해결 학습 모형 검증

본 모형의 적절성을 검증하기 위하여 모형 적절성 설문 문항의 내용 타당도 검사(1차)와 모형의 적절성 검사(2차)를 실시하였다.

1차 모형 적절성 설문 문항의 내용 타당도 검사는 컴퓨터교육분야의 교육전문가를 대상으로 각 문항별 리커드 3점 척도 '필수적임', '보통임', '필요없음'으로 구성하였고 CVR(항목내용타당도비율) 검증을 통해 결과를 분석하였다. 교육전문가의 교육 경력은 각각 20년 이상(8명), 15년 이상(1명), 5년 이상(1명)이다. 학력은 박사(3명), 석사(6명), 학사(1명)이다. 검사 결과 CVR 값이 8.0 이상인 문항을 최종 선정하였다.

2차 모형의 적절성 검사는 최종 완성된 설문 문항을 컴퓨터교육분야의 교육전문가를 대상으로 리커드 5점 척도 '매우그렇다'(5점), '그렇다'(4점), '보통이다'(3점), '그렇지않다'(2점), '매우그렇지않다'(1점)로 평가하였다. 교육전문가의 교육 경력은 각각 20년 이상(4명), 15년 이상(3명), 10년 이상(2명), 5년 이상(1명)이다. 학력은 박사(4명), 석사(4명), 학사(2명)이다.

평가 결과, 전체 문항 평점은 4.51로 긍정적인 것으로 나타났다. 특히, 학습 단계 구성(4.7)과 모형의 적용 효과 측면인 프로그래밍 학습에 도움(4.7), CT 증진에 도움(4.7)에서는 높은 평가 결과가 나타났다. 이는 모형에서 제공된 예제 확인을 통하여 학습 단계별로 문제를 해결하는 과정과 프로그래밍 과정이 실제 학습에 유용하고 컴퓨팅 사고력 구성 요소 발달에 긍정적인 효과가 있을 것으로 생각하는 것으로 해석할 수 있다.

<표 7> 2차 전문가 평가 평점 결과

문항 내용	평점
1. CT-TDPS 모형의 개발 전략은 적절하다.	4.6
2. CT-TDPS 모형의 학습 단계 구성은 적절하다.	4.7
3. '문제 정의 및 추론'의 하위 단계 구성과 진행 절차는 적절하다.	4.5
4. '테스트 코드 구현'의 하위 단계 구성과 진행 절차는 적절하다.	4.3
5. '실행 코드 구현'의 하위 단계 구성과 진행 절차는 적절하다.	4.3
6. '실행 코드 개선'의 하위 단계 구성과 진행 절차는 적절하다.	4.5
7. CT-TDPS 모형 단계별 산출물은 적절하다.	4.5
8. CT-TDPS 모형 단계별 관련 CT 연결은 적절하다.	4.3
9. CT-TDPS 모형은 프로그래밍 학습에 도움이 될 것이다.	4.7
10. CT-TDPS 모형은 CT 증진에 도움이 될 것이다.	4.7
평점	4.51

반면, 테스트 코드 구현(4.3), 실행 코드 구현(4.3), CT 연결의 적절성(4.3)에서는 상대적으로 낮은 평가 결과가 나타났는데, 이것은 테스트 중심의 프로그래밍에 익숙하지 않은 점과 그로 인해 컴퓨팅 사고력 구성 요소와의 연관성에 의구심을 갖는 것으로 추측된다. 학습에 앞서 모형에 익숙해지기 위하여 충분한 예제와 설명을 제공할 필요가 있을 것으로 생각된다.

6. 결 론

4차 산업혁명의 핵심 기술들은 우리의 미래의 모습을 바꿔놓고 있다. 그러한 핵심 기술들에는 소프트웨어가 중요한 부분을 차지한다.

소프트웨어를 읽고 쓰고 만들 수 있는 능력은 과거의 리터러시 교육처럼 소프트웨어 리터러시로 모든 사회 구성원이 알아야 할 기본 능력으로 간주되고 있고 전 세계적으로 소프트웨어 교육이 확산되고 있다. 이러한 추세에 발맞추어 우리나라도

2015년 개정 교육과정을 통해서 초중등 교육과정에 소프트웨어 교육을 강화하였다.

소프트웨어 교육은 컴퓨터 과학의 기본적인 개념과 원리를 통해 문제를 해결하는 것을 목표로 한다. 이 과정을 통해서 컴퓨팅 사고력이라고 하는 핵심 능력이 발현되고 향상되도록 지도한다. 컴퓨팅 사고력은 문제를 인간이 아닌 컴퓨터 관점에서 해결하는 과정을 통해서 향상될 수 있다. 특히, 문제해결 프로그래밍을 통해서 컴퓨팅 사고력을 구성하는 요소들의 향상 정도가 크다고 할 수 있다.

지금까지 문제해결 프로그래밍 교수 학습 모형의 대부분은 문제의 해결책을 찾고 이를 프로그래밍하는 과정으로 진행된다. 이는 전통적인 프로그램 개발 방식인 폭포수 모형(분석-설계-구현-테스트)을 따른다. 이러한 모형은 문제의 해결책이 올바르고 설계와 구현이 올바르게 전개된다면 결과도 올바르다고 간주할 수 있다. 하지만, 선행 단계에서 오류가 발생하였을 경우에 오류를 인지하지 못하고 마지막 단계인 테스트 단계에서 오류를 발견하는 경우가 대부분이다. 이럴 경우 문제의 해결책을 다시 찾고 구현하는데 적지 않은 시간과 노력이 발생하고 이는 학습에 부담으로 작용할 수 있다.

본 연구에서 제안하는 학습 모형은 문제해결 프로그래밍의 학습 절차에 애자일(Agile) 개발 방식인 테스트주도개발(TDD)을 적용한 것으로 문제 해결을 위한 테스트 코드를 먼저 작성함으로써 명확한 개발 목표를 설정하고 즉각적인 피드백을 제공해 설계상의 오류나 문제점을 프로그래밍 과정에서 바로 확인할 수 있도록 하여 학생들로 하여금 문제해결에 자신감을 갖게 하고 프로그래밍 학습에 몰입할 수 있도록 하였다. 또한, 각 학습 단계는 컴퓨팅 사고력 구성 요소에 기반하여 구성하였고, 실제 문제 해결 예제를 통하여 문제해결 프로그래밍 과정을 시연하고 전문가를 통하여 모형의 적절성을 검증하였다.

모형의 적절성 검증 결과, 대부분의 문항에서 긍정적인 평가 결과가 나타났다. 특히, 학습 단계 구성, 프로그래밍 학습에 도움, CT 증진에 도움 등의 문항에서는 평가 결과가 높게 나타났는데 이는 해당 모형의 학습 단계가 적절하고 실제 학습에 적용했을 때 문제해결 프로그래밍 학습과 컴퓨팅 사고력 발달에 긍정적인 효과가 있을 것으로 판단한 것으로 해석할 수 있다.

반면, 테스트 코드 구현, 실행 코드 구현, CT 연결의 적절성 등에서는 상대적으로 낮은 평가 결과가 나타났는데 이는 테스트를 먼저 만들고 프로그래밍을 한다는 점에서 기존 모형들과는 익숙하지 않은 점이 있고 그로 인해 컴퓨팅 사고력 하위 요소와의 연관성에도 의구심을 갖는 것으로 추측된다.

향후 본 연구에서 제안하는 컴퓨팅 사고력 기반 테스트 중심 문제해결(CT-TDPS) 학습 모형을 실제 문제해결 프로그래밍 수업에 적용하여 컴퓨팅 사고력 발달 등 학습에 미치는 영향에 대한 연구가 필요할 것으로 생각된다.

참 고 문 헌

[1] 이철현 (2016). 소프트웨어 교육을 위한 컴퓨팅 사고력 기반 문제 해결 모형(CT-PS Model) 개발. **실과교육연구. 22**(3), 23-44.

[2] 전경란 (2015). **미디어 리터러시의 이해**. 커뮤니케이션북스.

[3] 교육부 (2015). 실과(기술·가정)/정보과 교육과정. 교육부 고시 제2015-74호 [별책 10].

[4] 신정호, 박상오, 이규일, 전우균, 조건희 (2014). **TDD 이야기**. 한빛미디어.

[5] 오정철, 김종훈 (2016). 초등학생의 컴퓨팅 사고력 신장을 위한 퍼즐 기반 컴퓨터과학 수업 모형 및 프로그램 개발. **수산해양교육연구. 28**(5), 1183-1197.

[6] 전용주 (2017). **새로운 교육과정의 소프트웨어 교육을 위한 컴퓨팅 사고력 기반 창의적 문제해결(CT-CPS) 수업모형의 개발 및 적용**. 박사학위논문. 한국교원대학교.

[7] 김현배 (2018). 컴퓨팅 사고력을 기반으로 하는 정보교과 교수학습 모형 고찰. **정보교육학회논문지. 22**(1), 1-8.

[8] 김성식, 김영직, 조아라, 이민우 (2019). 문제해결 프로그래밍 교육에서 컴퓨팅 사고력 평가를 위한 도구 개발: 지필형 검사지 및 자기보고식 설문지. **컴퓨터교육학회 논문지. 22**(3), 89-99.

[9] 배학진, 이은경, 이영준 (2009). 문제 중심 학습을 적용한 스크래치 프로그래밍 교수 학습

모형. **컴퓨터교육학회 논문지. 12**(3), 11-22.
[10] 김영직, 안성훈 (2017). TDD를 활용한 문제 중심 프로그래밍 교육에 대한 고찰. **한국창의정보학회 하계학술발표논문집**. 61-66.
[11] 전석환, 김정동, 백두권 (2009). 테스트 주도 개발을 위한 유연한 단위 테스트 도구. **정보과학회 논문지:컴퓨팅의 실제 및 레터. 15**(2), 140-144.
[12] 채수원 (2010). **테스트주도개발 TDD 실천법과 도구**. 한빛미디어.
[13] Kayongo, P. (2016). *Why do software developers practice test-driven development?*. University of Cape Town.
[14] Edwards, S. H. (2003, August). Using test-driven development in the classroom: Providing students with automatic, concrete feedback on performance. In *Proceedings of the international conference on education and information systems: technologies and applications EISTA* (Vol. 3). Citeseer.
[15] Buffardi, K., & Edwards, S. H. (2013, March). Impacts of adaptive feedback on teaching test-driven development. *In Proceeding of the 44th ACM technical symposium on Computer science education* (pp. 293-298). ACM.
[16] Computing at School (n.d.). *Computational Thinking - How do we think about problems so that computers can help?.* Available from https://community.computingatschool.org.uk/
[17] Janzen, D. S. (2005). Software architecture improvement through test-driven development. *In Companion to the 20th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications* (p. 240).

김 영 직

1997 한국교원대학교 컴퓨터교육과(교육학학사)
2007 고려대학교 컴퓨터정보통신대학원 소프트웨어공학(공학석사)
2004 ~ 2016 한국교육개발원 이러닝센터 근무
2014 ~ 현재 한국교원대학교 컴퓨터교육학과 박사과정
2017 ~ 현재 경인교육대학교 시간강사
관심분야: 소프트웨어교육, 소프트웨어공학, 이러닝
E-Mail: capbang93@gmail.com

김 성 식

1977 고려대학교 경영학과(경영학사)
1978 ~ 1991 교육부 및 교육정책자문위원회 근무
1988 미국오리곤주립대학교 대학원 컴퓨터과학과(이학석사)
1992 고려대학교 컴퓨터학과(이학박사)
1992 ~ 현재 한국교원대학교 컴퓨터교육과 교수
관심분야 : 교육용 콘텐츠, 알고리즘, 원격교육
E-Mail: seongkim@knue.ac.kr