International Journal of Advanced Science and Technology
Vol.114 (2018), pp.35-48
http://dx.doi.org/10.14257/ijast.2018.114.04

Software Education Model for Non-major Undergraduate Students using DBSEM

Kil Young Kwon[1] and Won-Sung Sohn[2*]

[1]Dept. of Family Medicine, Eulji General Hospital, Hagye-Dong 280-1, Hanguelbiseok-street 68, Nowon-Gu, Seoul, 139-872, Korea
[2]Dept. of Computer Education, Gyeongin National University of Education, Incheon, Korea
**sohnws@ginue.ac.kr*

Abstract

In general, block-based programming tools have been used effectively in basic programming courses for undergraduate undergraduates [4-5]. Despite the advantages of block-based programming tools, however, unexpected issues have arisen in the process of applying this to untrained college students. In the educational field where the philosophy and purpose of the programming education are not understood accurately, the purpose of the script coding transformed into the learning of the contents Author ware rated with scenario implementation and the game design.

In this study, we propose a design based software education model (DBSEM) to provide optimized programming education to non - traders in order to solve this problem. In this model, we define core modules and concept modules that are the core of computational thinking and include multi-level teaching and learning strategies consisting of prototype design and coding exercises using a design thinking strategy. As a result, non-major undergraduates can acquire basic knowledge of computer programming more easily and can develop core competence in computing thinking in the course of programming using block-based coding tools. In addition, this study developed a curriculum based on DBSEM model and includes core module based conceptual learning, UX based prototype design, and block- based scripting (coding) practice based on 15 weeks' curriculum.

The proposed model has been applied to 312 non-major undergraduate students for the past 8 years. The analysis result of the proposed evaluation tool developed by the present research team has resulted in the final stage 3 level (good). As a result, non - technical undergraduates were able to shape the concept of programming fundamentals more firmly and to develop core competencies of computing thinking through programming practice. The proposed research method can easily apply to any adult who lacks the athletic knowledge of computer science.

***Keywords:** SW Education, Programming, Scratch, Code Quality, Design thinking*

1. Introduction

In 2015, the revised elementary and secondary curriculum has designated SW education as an essential subject in the secondary education field since 2018, and interest in programming education is increasing in the field of actual education. In addition, computing education for new undergraduates is spreading at home and abroad [1-2].

Received (November 25, 2017), Review Result (January 27, 2018), Accepted (February 4, 2018)
[*] Corresponding Author

In general, block-based programming tools [3] have been used effectively in basic programming courses for undergraduate undergraduates [4-5]. Despite the advantages of block-based programming tools, however, unexpected issues have arisen in the process of applying this to untrained college students. In the field of education where the philosophy and purpose of programming education are not understood accurately, the purpose of script coding transformed into the learning of contents authoring tools such as tool learning and scenario making and game authoring using it [6].

Also, top-level block instructions, which are characteristic of block-based programming tools, are represented by visual attributes of programming elements. Therefore, to improve computational thinking [7-13], which is the purpose of traditional programming education, it is necessary to surface the function of abstracted blocks.

In this study, we propose a design based software education model (DBSEM) to provide optimized programming education to non - traders in order to solve this problem. In this model, we define core modules and concept modules that are the core of computational thinking and include multi-level teaching and learning strategies consisting of prototype design and coding exercises using Design Thinking strategies. As a result, non-major undergraduates can acquire basic knowledge of computer programming more easily and can develop core competence in computing thinking in the course of programming using block - based coding tools. In addition, this study developed a curriculum based on DBSEM model and includes core module based on conceptual learning, UX based prototype design, and block- based scripting (coding) practice based on 15 weeks curriculum.

The proposed model has been applied to 312 non-major undergraduate students for the past 8 years. The analysis result of the proposed evaluation tool developed by the present research team has resulted in the final stage 3 level (good). As a result, non - technical undergraduates were able to shape the concept of programming fundamentals more firmly and to develop core competencies of computing thinking through programming practice. The proposed research method can easily apply to any adult who lacks the athletic knowledge of computer science.

2. Related Works

The effect of programming education is to promote computational thinking and has been widely spread by Professor J. Wing's advocacy. Compared to the theory related to the existing programming education, Prof. J. Wing's differentiating definition is that 'computing thinking' is defined not as the thinking power needed for IT majors but as literacy essential for every kind of elementary and middle school students like reading and writing, And the direction of the future. The practice of defining and disseminating computational thinking has recently been very diverse. In particular, the inclusion of Computing as a regular course in k-12 in the UK in 2015 has also had a major impact on the United States and Korea. At the center of these various efforts, the development of scratch, a block-based coding tool, has had a great impact [1,11].

Block-based programming tools based on 'Tinkering' [14-15] or puzzle metaphor [16] abstract the programming grammar to the top level and transform it into a script using visual cues called blocks. Since the command blocks abstracted at the top level are executed only in the combination of the same attributes (time, function, *etc.*), unlike a general programming language, grammatical errors related to code generation do not occur originally. Blocks the functional classification of blocks based on color attributes and the clarification strategy of logical relations using visual attributes of blocks provide a recognition process and an affordance effect that is far superior to existing programming languages, it is possible to write [16]. In addition, the functional classification of blocks based on color attributes and the clarification strategy of logical relations using visual attributes of blocks provide a recognition process and affordance effect, which are much higher than existing programming languages, so that scripts can be written at various ages. [16]. Alice [5], Scratch [14], Snap! Block-based programming tools such as [17] and Blockly [18] are actively used not only as introductory lectures for undergraduate students [2] but also for beginners' coding introductory education [11,15] And has contributed greatly to the popularization of SW education.

On the other hand, the biggest difficulty in the block-based programming education field is to derive the computational thinking ability items of the blocks embedded in the interface, and various studies have conducted to define and evaluate them. In general, we quantify the number of significant blocks created by scratches to evaluate the quality of programming output or suggest a script evaluation rubric related to design script, and interface. Michal Armoni and Moti Ben-Ari [19,20] analyzed the concept of computer science contained in Scratch in eight items and developed test items and evaluation items for qualitative evaluation in order to evaluate conceptual understanding of each area.

3. Designed based Software Education Model

In this study, we developed a design-based teaching-learning model (DBSEM) to apply effective software education to undergraduate students as shown in Figure 1.

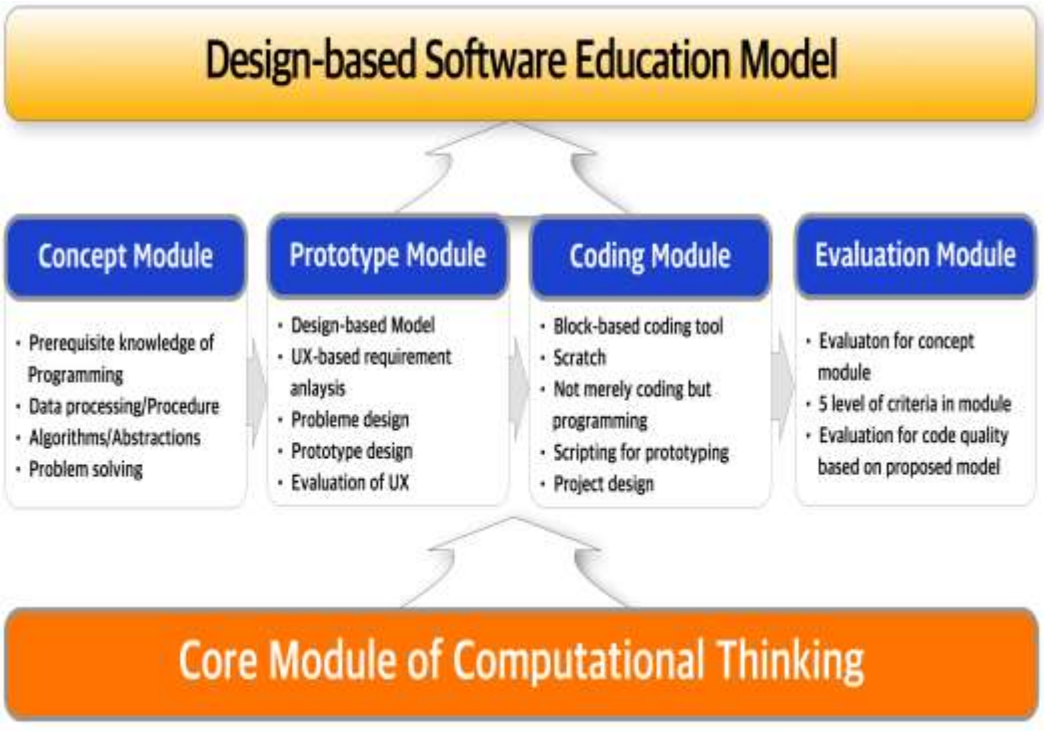

Design-based Software Education Model
Concept Module
Prerequisite knowledge of Programming
Data processing/Procedure
Algorithms/Abstractions
Problem solving
Prototype Module
Design-based Model
UX-based requirement anlaysis
Probleme design
Prototype design
Evaluation of UX
Coding Module
Block-based coding tool
Scratch
Not merely coding but programming
Scripting for prototyping
Project design
Evaluation Module
Evaluaton for concept module
5 level of criteria in module
Evaluation for code quality based on proposed model
Core Module of Computational Thinking

Figure 1. Teaching-Learning Model for Software Education (DBSEM)

3.1. Computational Thinking Core Module

Computational thinking has been applied by Professor Seymour Papert in 1980 [21] and 1996 [22] and has been very effective in solving complex large-scale problems using computational thinking in various fields and improving efficiency. Professor J. Wing refers to computing thinking as a universal thinking and technology that not only computer scientists but also computer scientists can learn and use, and it is the basic literacy such as reading, writing, and computation. I am convinced. Computational thinking is based on the basic concepts of computer engineering, solving problems, designing systems, and securing the ability to understand human behavior fundamentally [7-8].

These thinking have had a major impact on the introduction of public education in the UK in 2014 and have led to major changes in educational policies in advanced countries such as the United States, France, China, and Japan. The above phenomenon has also had a great impact on the field of the university. Various efforts have introduced basic principles of programming and the basics of computer science to fresh undergraduates, especially non-major undergraduate students. In this process, it is desirable to form a conceptual knowledge so that non-technical undergraduate students can develop a computing thinking, and prepare a place for practical training.

In this study, we define a core module to improve computing thinking and consider it as a conceptual reference point of the proposed teaching-learning model as like below Table 1.

Table 1. Concepts and Definitions of Computation Thinking in DBSEM's Core Module

Concept	Description
Abstraction	Identifying and extracting relevant information to define main idea(s)
Automation	Design heuristic methods to diminish repetitive tasks using their own algorithms
Algorithm design	Reformulating the problem into a series of ordered steps
Data analyze and management	Logically organizing and analyzing data to design with finding patterns or developing insights
Decomposition	Breaking the problem down into smaller parts
Parallelization	Identifying, analyzing, and implementing possible solutions with simultaneous processing for achieving efficient and effective steps and resources
Pattern analyze and recognition	Creating and observing models, rules, principles, or theories of observed patterns to test predicted outcomes
Problem solving	Approaching the problem using programmatic thinking techniques such as iteration, symbolic representation, and logical operations
Simulation	Developing a model to imitate real-world processes with prototype with Design-thinking methods

3.2. Concept Module

The key to computing thinking is not programming, but conceptualization. One of the topics of computer engineering's subtopics is computer programming. Thinking like a computer engineer requires a very abstract and conceptual thinking logic than computer programming [7-8]. Therefore, the ultimate goal of programming education should be to achieve the conceptualization of computing thinking. To convey this to non-technical undergraduates, it is necessary to define concrete and logical modules to achieve the educational purpose.

Table 2. Description of Programming Concept Module in DBSEM for Scratch

Items	Descriptions	Scratch's Blocks
Command	-Block stacking strategies to make command of script	Most cases
Parameter	-Understanding of argument to return value	Make a Block: Add number, string, Boolean, text
Variable	-Value, reference, scope	Make a Variable
Trigger	-Event handling	Events blocks: when, broadcast
Conditional Statement	-If then, If then else	Control: if then else
Iteration	-Repeat, forever, wait until, repeat until	Control: repeat until
Data Type	-Boolean, number, string	Weakley-typed script language
Input &Output	- keyboard/Mouse Input/output, Designing Interactions	When key pressed
Data Structure	-Data control using List	Make a List
Concurrency	-Understanding of multi-threading using event handler	Broadcast
Procedures	-Making subroutine, Reusing of functions, Controlling of parameters using Make a Bloc	Make a Block, define

In this study, we defined a concept learning module based on the 'core module' described above. It does not at simply technical education of programming, but it aims at acquiring basic concepts of computing thinking through programming, designing strategy for problem- solving, and finally, developing capability as implementation result. Especially, this study assumes scratch based learning, a block - based coding tool, and defines three modules: a programming module, a complex module, and a functional module. The programming module intended to apply the programming elements contained in the scratch to the teaching and evaluation process, and to prevent the block-based coding tool from misrepresenting the content authoring training rather than programming education. Table 2 shows the results.

Computational thinking does not aim at learning simple programming skills, but it is ultimately purposed at acquiring the ability to define the problem and to find the best solution by performing optimal algorithmic thinking to solve it. This goal embodies in the process of designing the abstraction strategy for problem solving by applying the above-mentioned programming conceptual elements as a foreground and applying it in a complex way. To do this, we define the complex module to solve the following problem-solving ability as shown in the following Table 3.

The final step in the conceptual learning process is to derive a project-based output. In this process, it is essential that the learning process to form and conceptualize computing thinking and the development process to embody it are essential. The results of the development include algorithms for problem definition and resolution based on programming thinking, as well as fusing elements such as creative scenario design and aesthetic capabilities and functionality. In the present study, these factors are defined as functional modules as shown in the following Table 4 and proposed as detailed modules of concept learning.

Table 3. Description of Composited Concept Module

Items of composited module	Descriptions
Algorithms	- problem solving strategies with computer algorithms - create several conclusions
Divide & Conquer	- divide of instance with top-down/bottom-up techniques for problems solving
Interaction Design	- interaction design with dynamic data input, attribute of sprite

Table 4. Description of Functional Module

Items of functional module	Descriptions
Creativity	- level of contents story and scenario
Aesthetics	- multimedia functions and customizing of scratch object for the aesthetics
Functionality	- functionality of contents
Level of completion	- story structure with several path of scenario - options for user selection

3.3. Prototype Design Module

In addition to theory learning, design and development processes are required to deepen understanding of concept learning. However, it is not desirable to demand or expect a high level of learning because the level of exposure to basic computing theory is very low for undergraduate students. At the same time, deploying students to the process of developing a coding project in this environment can further degrade their immersion and interest in learning. In reality, it is a reality that such problems are frequently occurring in subjects such as programming and CS 501 that operated in college liberal arts [6].

This module based on the widely used models in design thinking and HCI, and can used to verify the hypothesis by designing and developing problem analysis and requirements. Especially, a strategy can implement its algorithm easily and efficiently by using paper prototype or prototype design tool based on low-fidelity without using difficult programming or coding tools in problem analysis and algorithm implementation. A prototype generates a model during the initial development of a system or development result. It is not perfect but used very effectively in the logic design process of the system. Therefore, this strategy intends to promote more competence to conceptualize and abstract the basic theories of computing thinking expressed in conceptual learning and to provide pre-learning functions of full-scale coding practice. It is possible to provide advantages of the following example shows a low-level low-fidelity prototype as shown in Figure 2 and a high-fidelity prototype tool as shown in Figure 3 for the topic Understanding Interactive Programming.

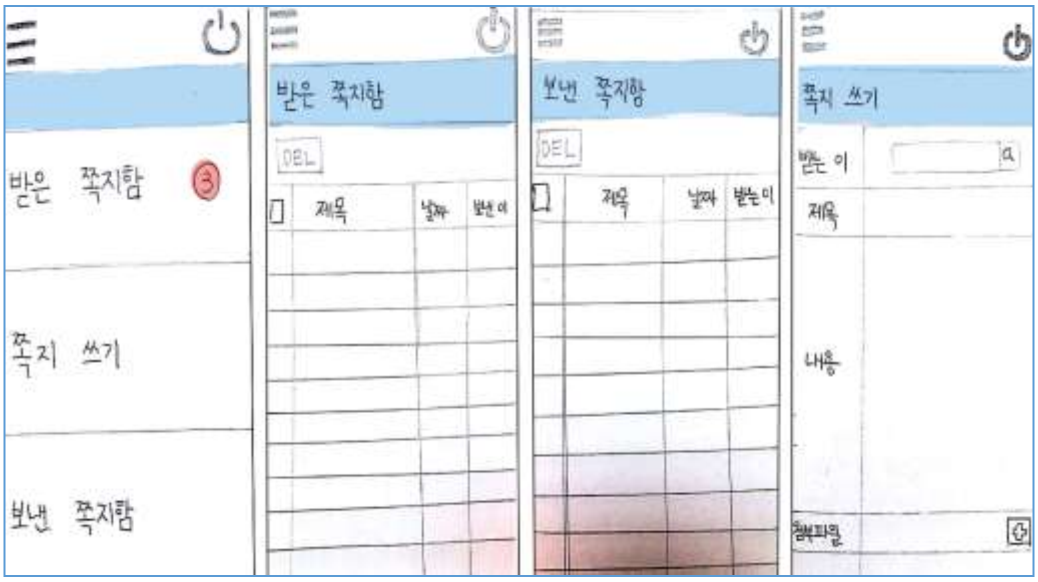


Figure 2. An Example of Low-fidelity Prototype Design for Interactive Programming

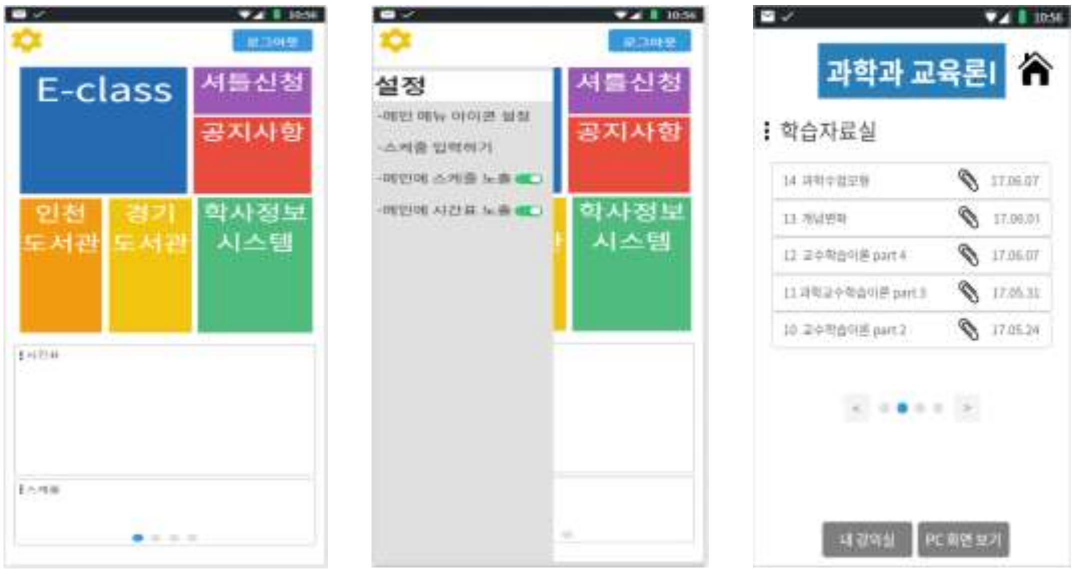


Figure 3. An Example of Hi-fidelity Prototype Design for Interactive Programming

3.4. Coding Module

In this study, the student can define problems through the prototype design module, specify their own algorithm, and establish the concept. Therefore, the algorithmic thinking implemented in the concept formulation phase transformed from the actual coding process to the programming domain. In this paper, we propose a coding module as shown in Figure 4.

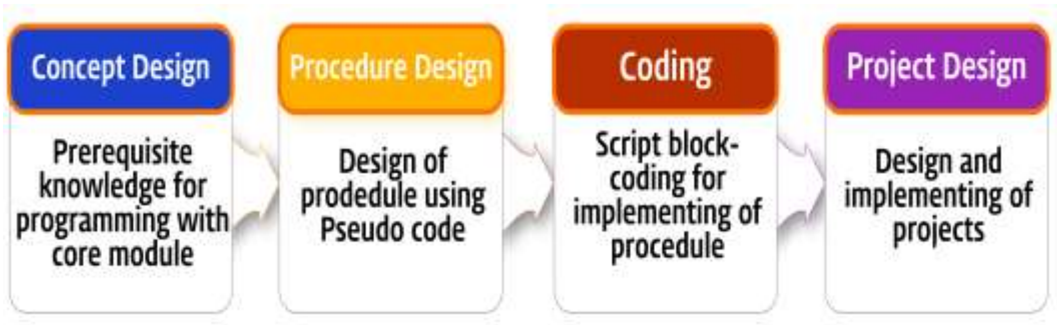
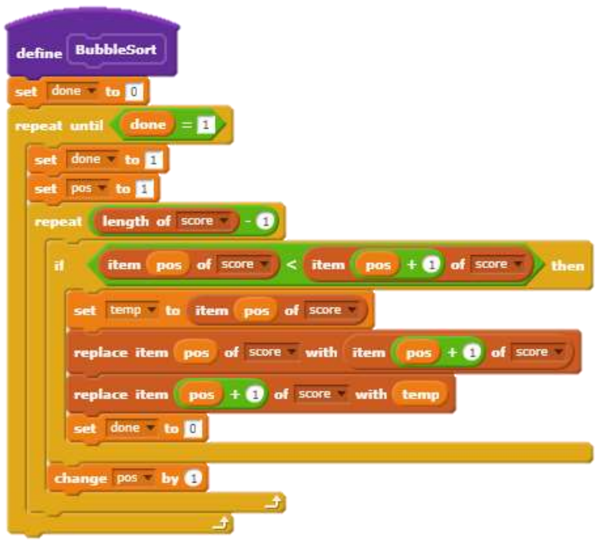

Concept Design
Prerequisite knowledge for programming with core module
Procedure Design
Design of prodedule using Pseudo code
Coding
Script block-coding for implementing of procedule
Project Design
Design and implementing of projects

Figure 4. Prototype Design Module for Concept Knowledge of Computing

In the proposed module, we first developed the basic theory embedded in the core module, thereby activating the computing accident and establishing the knowledge learned in the prototype module more firmly.

In order to refine the fractional knowledge of the computational thinking formed through this essential concept learning process, this step needs to implement a small-scale procedure based on pseudo code for each subject, and based on the contents of the detailed module of concept learning do. Figure 5 shows an example of numerical code learning and scratch coding (scripting) for bubble alignment.

```
procedure bubbleSort(A:list of
sortable items) defined as:
  for each i in 1 to length(A) do:
       for each j in length(A) down to i
+ 1 do:
         if A[ j ] < A[ j - 1 ] then
           swap( A[ j ],  A[ j - 1 ] )
         end if
       end for
  end for
end procedure
```



```
define BubbleSort
set done to 0
repeat until done = 1
  set done to 1
  set pos to 1
  repeat length of score - 1
    if item pos of score < item pos + 1 of score then
      set temp to item pos of score
      replace item pos of score with item pos + 1 of score
      replace item pos + 1 of score with temp
      set done to 0
    change pos by 1
```

Figure 5. A Case of Scripting for Psuedo Code using Scratch: Case of Bubble Sort

A pseudo- code is a way of describing an algorithm in a common language without following the grammar of a particular programming language, and it is readily available to a novice who is not familiar with the programming language. The ultimate goal of the practice stage is to define the problem using the knowledge that formed through the basic concepts and practices that have established through the theories and to present and implement the ideas to solve them. To do this, the prosoed model is to carry out small-scale (two or three subject-based) projects and long-term projects for the semester.

3.5. Evaluation Module

In the proposed model, to evaluate the level of the block script, we developed the basic concept of programming, the complex concepts about the algorithm and the problem- solving strategy, and the evaluation items related to the functional elements (104 items in total). Based on these evaluation items, the block script is evaluated according to the evaluation criteria shown in the following *Table 5*. After completion of the evaluation, the level of the final script is evaluated according to the script-level calculation model developed by the present inventor [6].

Table 5. Evaluation Criteria of Code Quality

Score	**1 Point**	**2 Point**	**3 Point**	**4 Point**	**5 Point**
Definition	Lack of understanding basic concept and level of implement	Understanding basic programming concept	Be above the average level of understanding basic concept and implement	Excellent in understanding of concept and implement	Fully understanding of concept and implement
	Vulnerable	Poor	Fare	Good	Excellent

In this study, the score calculated by reflecting the weight on the total evaluation value derived from 104 evaluation items convert into a percentage of the scale of the evaluation index, and the level of the programming code (script) is determined according to the result. The criterion for evaluating the level according to the final score of such a summed script is shown in Table 6 below. In the proposed method, the evaluation score is assigned to the 5th step of the code level (5 points scale) (1 point: 1 point to 5 points: 5 points) 2 points include 40%, 3 points for 60%, 4 points for 80%, and 5 points for 100%. The level is determined according to the final evaluation result, and the status of the learner is objectively determined according to the level definition.

Table 6. Five Levels of Code Quality

Levels	**Score Range**	**Level describe**	**Level definition**
Level 1	Score < 20	Vulnerable	- block composition and basic interaction design
Level 2	20 ≤ Score <40	Poor	- data processing/control, complicate interaction/ event design
Level 3	40 ≤ Score <60	Fare	- ability of data structure and procedure design
Level 4	60 ≤ Score <80	Good	- understanding algorithms for problem solving and designing of complicate interaction
Level 5	80 ≤ Score <100	Excellent	- fully understanding basic and complicate concept of programming

4. Curriculum Development and Application

4.1 Curriculum Development

In this study, we developed the curriculum as shown in Table 7 to perform programming education for undergraduate students. This course consists of basic theory education for concept learning formation, prototype design for theory practice, and scripting training course using scratch based on DBSEM. In the conceptual learning stage, we provide basic knowledge of computer science and programming that plays the role of athletic knowledge. The degree of difficulty can adjusted variously according to the object. In the prototype stage, we understand the algorithm for the project that will implemented during the hands-on learning process, design, and implementation of low-level prototype for this. Finally, in the course of the lab, we provide an experience to materialize core module theories by expressing algorithms using scratches and designing various interactions.

4.2. Curriculum Application and Analysis

Table 7. Curriculum of Software Education for Non-Major Undergraduate Students

Weeks	subject		
	Concept learning	Prototype design	Practice
1st	- HCI 101 - Programming Language 101	- Understand command - Basic of algorithms design	- Basic of Scratch - Understand UI
2st	- understand of binary - computation of binary	- design of binary card game	- practice of computational - Design of ping-pong game
3st	- Introduction of multimedia - Understand of object	- Geometric pattern design - Polygon pattern design	- Variable - Motion drawing
4st	- Animation theory 12 - Understand of MIDI	- Paper prototype - Cel animation	- Sprites - Looks/Sound block
5st	- Understand of trigger - Event programming	- Trigger based game design - event, condition, action	- Event block - Understand of UI
6st	- Understand of data - Understand of variable	- code/decode - Value, reference, scope	- Variable block - Science simulator project
7st	Midterm exam		
8st	- Basic of procedure - top-down/bottom-up	- top-down/bottom-up prototype	- Make a block - House with procedure
9st	- Problem solving with programming	- Algorithms with flowchart	- Conditional, Repeat - Line Tracer project
10st	- Understand of Reclusive - Understand of Threading	- Fractal with reclusive programming - Nesting/Recursion prototype	- fractal scripting - broadcast block
11st	- String processing - Interactive programming	- string prototype - Quiz UI	- Quiz project - Hangman project
12st	- Understand of data structure I	- Searching prototype	- List block - Linear search project
13st	- Understand of data structure II	- Sorting prototype	- Bubble sort - Median searching
14st	Final project		
15st	Final exam		

In this study, we proposed a teaching - learning model applicable to non-major undergraduate students using block-based programming tool and develop a curriculum based on this. The validity of the proposed model is verified by applying the non-major students in this study to the University of Education.

The purpose of this study is to investigate the effect of the DBSEM model - based curriculum of 312 students who took the 'Practical Practice' course of the 3rd year students of the College of Education between 2008 and 2015. The results are shown in Table 8 below. Details of the level measurement tool [6] are omitted in this paper.

Table 8. Evaluation and Analysis of Script Quality for the Final Scores of Students

Category	Evaluation Criteria						Level of criteria	Level of Group
	Sub Criteria	Weight(A)	Score(B)	Weighted Score(AxB)	Code Quality	Quality Level		
Programming Concept	Command	0.013	12	0.156	54.67	3	54.77	54.46 Level 3
	Parameter	0.007	10	0.070	53.45	3		
	Variable	0.051	7	0.357	51.69	3		
	Trigger	0.041	8	0.328	59.02	3		
	Conditional Statement	0.090	11	0.990	55.84	3		
	Iteration	0.088	12	1.056	59.81	3		
	Data Type	0.028	12	0.336	69.16	4		
	Input&Output	0.038	10	0.380	58.84	3		
	Data Structure	0.053	7	0.371	47.21	3		
	Concurrency	0.055	7	0.385	53.39	3		
	Procedures	0.059	5	0.295	39.35	2		
Advanced Concept	Algorithms	0.130	6	0.780	41.86	3	46.02	
	Divide&Conquer	0.128	6	0.768	39.11	2		
	Interaction Design	0.073	10	0.730	57.10	3		
Contents Evaluation	Creativity	0.044	10	0.440	58.98	3	59.95	
	Ascetics	0.029	9	0.261	57.22	3		
	Functionality	0.029	11	0.319	67.75	4		
	Level of Completion	0.044	10	0.440	55.84	3		
Sum		1	163	8.462	54.46	3		

Table 8 shows the results of applying the measurement model of programming level to the results of the curriculum based on the DBSEM model of the proposed study. The weighted conversion score of 8.462 for the final score of 312 final scores applied to the study was 54.46% of the script level and the result of the final stage 3 level (good) was derived. When analyzing the results of the students in detail, all items except for the 'procedure' item that recorded at least 2 level (insufficient) among the concepts of the core module were evaluated as 3 level (good). The concept of Procedure is not only a technique for writing functions but also a core concept for algorithm development for problem- solving. These results show that the results of studying the detailed concepts of the divide & conquer and algorithms fields of the complex module affect the results. Especially, the concepts related to the problem- solving strategy (level 2) should be more improvement later.

5. Conclusion

In this study, we proposed a design based software education model (DBSEM) for programming education for the non-major student. The proposed model defines 'core module' and 'concept learning' module, which are the core of computational thinking ability, and includes prototype design and coating practice based on a design thinking strategy. In addition, this study developed a curriculum based on the proposed model and consists of concept learning process, prototype design and coding practice based on 15 weeks curriculum.

In this paper, we applied the DBSEM - based curriculum to 312 non-major undergraduate students and analyzed through the evaluation tool developed by our researchers, and the final stage 3 level (good) results show. As a result, non-major students were able to shape the basic concepts of programming through the proposed curriculum and to plan core competencies of computing thinking through programming practice.

The DBSEM model can be applied effectively for establishing computing incidents in various fields required in the era of the rapidly changing fourth industrial revolution. The key concept learning and prototype practice strategies of the proposed study are easy to learn for any adult who lacks a player knowledge of computer science, so that he/she can learn not only programming education but also various fields of SW education such as' physical computing 'and' It can be applied and applied effectively. In the future, this study will be supplemented with studies on qualitative satisfaction analysis or learning effect.

Author Contributions

Kil Young Kwon designed experimental model for this research.

Acknowledgments

This paper is a revised and expanded version of a paper entitled ‘Design based software education model for Non-major Undergraduate Students’ presented at 2017 1st International Workshop on Cultural and Technological Exchange and Mutual Development of The Pacific Rim Countries (CuTEMD 2017), December 17, 2017, Namseoul University, Cheonan, Korea.

References

[1] D. J. Malan and Henry H. Leitner, “Scratch for budding computer scientists”, SIGCSE Bull, vol. 39, no. 1, **(2007)**, pp. 223-227.

[2] S. Mishra, S. Balan, S. Iyer and S. Murthy, “Effect of a 2-week scratch intervention in CS1 on learners with varying prior knowledge”, In Proceedings of the 2014 conference on Innovation & technology in computer science education (ITiCSE '14), ACM, **(2014)**, pp. 45-50.

[3] D. Weintrop, “Minding the Gap Between Blocks-Based and Text-Based Programming”, Proceedings of the 46th ACM Technical Symposium on Computer Science Education, **(2015)**, ACM.

[4] C. Hundhausen, S. Farley and J. Brown, “Can direct manipulation lower the barriers to computer programming and promote transfer of training?: An experimental study”, ACM Trans. Computer-Human Interaction, vol. 16, no. 3, **(2009)**.

[5] I. Utting, S. Cooper, M. Kölling, J. Maloney and M. Resnick, “Alice, Greenfoot, and Scratch - A Discussion”, Trans. Comput. Educ., vol. 10, no. 4, **(2010)**, Article 17.

[6] W. S. Sohn, “A Developing a Model for Measuring of Programming Education for Non-major Undergraduate Students”, Journal of The Korean Association of information Education, vol. 20, no. 3, **(2016)**, pp. 293-302.

[7] J. M. Wing, “Computational thinking”, Communications of the ACM, vol. 49, no. 3, **(2006)**, pp. 33-35.

[8] J. M. Wing, “Computational thinking and thinking about computing”, Philosophical transactions of the royal society of London A: mathematical, physical and engineering sciences, vol. 366, no. 1881, **(2008)**, pp. 3717-3725.

[9] V. Barr and C. Stephenson, "Bringing computational thinking to K-12: what is Involved and what is the role of the computer science education community?", ACM Inroads, vol. 2, no. 1, **(2011)**, pp. 48-54.
[10] S. Grover and R. Pea, "Computational thinking in K–12: A review of the state of the field", Educational Researcher, vol. 42, no. 1, **(2013)**, pp. 38-43.
[11] M. Resnick, J. Maloney, A. Monroy-Hernández, N. Rusk, E. Eastmond, K. Brennan and A. Millner, "Scratch: programming for all", Communications of the ACM, vol. 52, no. 11, **(2009)**, pp. 60-67.
[12] K. Brennan and M. Resnick, "New frameworks for studying and assessing the development of computational thinking", In Proceedings of the 2012 annual meeting of the American Educational Research Association, Vancouver, Canada, **(2012)**, pp. 1-25.
[13] M. Guzdial, "Education Paving the way for computational thinking", Communications of the ACM, vol. 51, no. 8, **(2008)**, pp. 25-27.
[14] J. Maloney, M. Resnick, N. Rusk, B. Silverman and E. Eastmond, "The Scratch Programming Language and Environment", ACM Transactions on Computing Education, vol. 10, no. 4, **(2010)**.
[15] W. Slany, "Tinkering with Pocket Code, a Scratch-like programming app for your smartphone", Proc. of Constructionism, Vienna, Aus, **(2014)**.
[16] R. Lister, "Computing Education Research: Programming, syntax and cognitive load", ACM Inroads, vol. 2, **(2011)**, pp. 21-22.
[17] B. Harvey and J. Mönig, "Bringing no ceiling to Scratch", Proc. of Constructionism 2010, Paris, Fr., **(2010)**, pp. 1-10.
[18] N. Fraser, Blockly, Google, **(2013)**.
[19] O. Meerbaum-Salant, M. Armoni and M. Ben-Ari, "Learning computer science concepts with Scratch", Computer Science Education, vol. 23, no. 3, **(2013)**, pp. 239-264.
[20] O. Meerbaum-Salant, M. Armoni and M. (Moti) Ben-Ari, "Learning computer science concepts with scratch", In Proceedings of the Sixth international workshop on Computing education research (ICER '10). ACM, New York, NY, USA, **(2010)**, pp. 69-76.
[21] P. Seymour, "Mindstorms: Children, computers, and powerful ideas", Basic Books, Inc., **(1980)**.
[22] P. Seymour, Int J Comput Math Learning, vol. 1, no. 95, **(1996)**.

Authors

Won-Sung Sohn, he received the B.S. and M.S. degrees in Computer Engineering from Dongkuk University in 1998 and 2000 and the PhD degree in Computer Science from Yonsei University in 2004. From 2004 to 2006. He was a postdoctoral associate in the Computational Design Laboratory at Carnegie Mellon University. He is currently a professor at Department of Computer Education, Gyeongin National University of Education. His research interests include educational design research, human-computer interaction and computer education.

Kil Young Kwon, she is currently an assistant professor in the Department of Family Medicine at Eulji General Hospital and Eulji University, Republic of Korea. She served as Internship, Resident & Fellow in the Severance Hospital at Yonsei University College of Medicine, Republic of Korea from March 2001 to February 2006. She received a M.S. International Health Science in the department of Public Health, Yonsei University and Ph.D. Integrated Medicine in the department of Medicine, Cha University, Seoul, Korea, in 2008 and 2015, respectively.