AN ADAPTIVE ALGORITHM FOR LEARNING COMPUTER PROGRAMMING COURSE

Deachrut Jaithavil

Department of Computer Engineering, Rajamangala University of Technology Thanyaburi, Thailand

Natha Kuptasthien

Department of Industrial Engineering, Rajamangala University of Technology Thanyaburi, Thailand

ABSTRACT

This paper aims to compare active learning and passive learning in a Computer Programming course for the 1st year engineering students. The CDIO standard 7 and 8 was implemented to change teaching methods. The students were divided into two classes. An active learning environment was provided for Class A, while Class B was offered a passive learning classroom environment. The passive learning included a lecture and computer-based materials. Meanwhile, the active learning class focused on designing activities that were suitable for the expected learning outcomes and whether students understood the concept behind programming. Active learning activities were designed to assure students' learning outcomes from remembering and understanding to applying the knowledge in computer programming. To develop a deeper understanding, the students practiced the algorithms using interactive programs. To improve the thinking process, visual block-based programming language in form of a jigsaw puzzle was introduced. Each specific block has a different color, which can be dragged together to build applications that creates different possible outcomes. Later on, the student applies their knowledge of programming languages to electronic devices that use sensors and microcontrollers, which translates analog input into a software system that controls electro-mechanical devices such as motors, servo, lighting or other hardware. This last phase has engaged students in applying, analyzing, and evaluating ideas with text-based programming language based on active experiential learning. Both classes were evaluated based on their pre-test and post-test performances. The independent sample t-test result found that the outcomes of Class A students were statistically significantly higher than the Class B students at the 0.05 level of significance. It encouraged the instructor to further develop the course, regarding the visual block-based programming language, the text-based programming language, problem-solving skills and other necessary skills.

KEYWORDS

Visual block-based programming, computer algorithms, computer engineering, standards: 7, 8.

INTRODUCTION

Extreme modifications in the tertiary education system require the university to improve the quality of education. Several new curriculums are designed to support a more diverse range of students. State-of-the-art infrastructures and technology are able to enhance learning experiences. However, the university pedagogy remains challenged, with most lecturers still use lecture-based practices. The assessment of student competency relies on how students solve exercises and textbook problems (Vega et al., 2013).

This situation also occurs in the Computer Programming course at the faculty of engineering, Rajamangala University of Technology (RMUTT), Thailand. This course is offered to all new 1st year students entering all of the engineering disciplines. The course covers computer concepts, computer components, hardware and software interaction, electronic data processing concepts, program design, development methodology and high-level language programming. The teaching team found that non-programming engineering students did not fully understand the content due to the increasing level of difficulty in recent years. The topics that the students struggled with the most with was program design and development methodology, problem-solving, and algorithm. The student's feedback results reveal that the main issues interfering with their learning were the heavy lectures with minimal activities to provide students experiences that shape their understanding of the content. Berglund and Persson (2018) stated a similar situation where computer programming is perceived as theoretical, abstract, and complicated with less connection to real-world application, especially for non-programming engineering students.

In order to solve the mentioned problems and encourage non-programming engineering students to gain a deeper level of understanding and achievable learning experience, the lecturer applies Integrated Learning Experiences (CDIO standard 7) and Active Learning (CDIO standard 8) techniques. Thus, this paper aims to:

- Design appropriate activities that support students learning experiences and increasing levels of interest in learning computer programming.
- Compare learning outcomes between an active learning and passive learning groups of 1st year non-programming engineering students

EARLIER WORKS

Computer literacy education becomes crucial for younger learners in this decade. Many primary and secondary schools worldwide integrate the knowledge of RFID cards, radar ranging, smart street lights, intelligent traffic lights, remote control, game programming, scratch programming and Arduino in educating young learners to experience basic computer programming (Yongqiang et al., 2018). Computational thinking is an essential problem-solving technique that involves logical, algorithmic processes and reasoning abilities. Computational thinking is regularly brought up in the context of learning computer programming. Wing (2006) has developed key principles of computational thinking, as shown below:

- Decomposition: divide the problems into a small portion
- Pattern Recognition: observe the similarities and differences of sequences, formats or steps
- Abstraction: select format, apply problem solving process, trial-and-error
- Algorithmic Thinking: create a solution with systematic problem-solving skills and reasoning.

There are several examples of literature that focuses on teaching and learning 1st year students and computer programming. Siong and Thow (2017) succeeded in raising students' motivation by using a "learning-by-doing" approach for the 1st year digital electronic course. The inquiry and reflection process allows the student to develop a better understanding of the concept. Deep learning in experimentation, discussion in seminar group, 3D-model software to develop physical products and programming exercises show a promising approach to motivate non-programming engineering students in the introductory 1st year course (Berglund and Persson, 2018). Shorn (2018) stated that the student found computer programming courses boring, time-consuming and difficult. Gamification, an application of gaming elements in a non-game context, was used. Positive results show that the methodology can support students' learning and gains more interest in learning computer programming.

Among many applications on teaching computer programming, Scratch-Arduino is a highly effective tool to teach logical thinking and creativity. The S4A (Scratch 4 Arduino) provides a high-level user interface with simple and interactive functions. Thus, the S4A platform is appealing to novice programmers (Gupta et al., Hladik et al., 2017; Roscoe et al., 2014; Tangney et al., 2010).

RESEARCH METHODOLOGY

The research question is "Is there any differences in computational skills between Class A (active learning with a visual block-based programming language) and Class B (passive learning with text-based programming language)?"

The author applies the Tyler model along with Behaviorism and Constructivism theories in designing the Computer Programming course. Tyler model (Tyler, 1967), is an essential theory of curriculum development in the scientific approach. With four steps:

1. Determine the objectives course or learning outcomes
2. Identify educational experiences related to the purpose
3. Organize the experiences
4. Evaluate the purposes

Nature of Course and Requirements

The Computer Programming course grants 3 credits to 1st year engineering students from different engineering disciplines and is mandatory for all engineering majors. The students are diverse in backgrounds, prior knowledge in programming, skills and interests. A semester contains 16 weeks of lessons, midterm and final examinations. Each week, the lesson comprises of a 2-hour lecture and 3-hour practical exercises. The normal class size is 40 students.

Participants

An active learning classroom environment was provided for Class A (an experimental group), while Class B (a control group) was offered a passive learning classroom environment. The active learning class focused on activities designed to be suitable for the expected learning outcomes and to check whether the student fully understands the concept behind programming. The passive learning environment included a traditional lecture and computer-based materials. The experiment was conducted using purposive sampling of registered

students in the course of semester 1 in the 2018 academic year (June - October 2018). Class A (an experimental group) had 39 students, while Class B (a control group) had 38 students.

Assessing Learning Achievement and Data analysis

An assessment tool was a test including 50 multiple-choice questions (50 points). The students' pre-test and post-test results were used to assess and determine the student learning achievement. The pre-test was conducted in week 2, while the post-test was in week 11 of the semester. The questions covered the program design and development methodology of algorithm concepts with flowcharts, which were validated by all the lecturers in the course. A quantitative analysis was performed using an independent sample t-test with a confidence level of 95%.

Intended Learning Outcomes

Intended learning outcomes (ILOs) are set as shown in Table 1. The students are expected to achieve these following outcomes after finishing this course.

Table 1. Learning outcomes for 1st year computer programming course

ILO1	To understand the concept of problem solving
ILO2	To understand steps in an algorithm development
ILO3	To understand the concept of an Algorithm
ILO4	To understand the concept of a Flowchart development

Designing a Course Syllabus

A 16-week course syllabus was designed, as shown in Table 2. The authors applied a Constructive Alignment theory (Biggs and Tang, 2007) to design classroom activities that focus on developing the student's logical and creative thinking skills, engineering reasoning and problem-solving skills. The designed activities must be aligned with the intended learning outcomes.

Table 2. Course Syllabus

Week	**Topics**
1 – 2	Introduction to Computer
3 – 6	Introduction to Problem Solving ● Procedure and Steps ● Algorithm ● Flow Chart ● Symbols used in Flow Charts ● Pseudo Code
7 – 8	Introduction to C Language
10 – 11	Control Structure
12 – 13	Function
14 – 15	Array
16	String

Teaching & Learning Activities

Teaching and learning activities focused on developing professional skills with knowledge construction rather than memorization. Table 3 shows 3 active learning activities offered to Class A (an experimental group)

Table 3. Active learning activities

Activity	Bloom Taxonomy	Week	Topics and Activity Details	Practice Hour
1	- Remembering - Understanding	3 – 6	Introduction to Problem Solving ● Use a Flowgorithm program ● Drag and drop flow chart symbols to the problems	8
2	- Understanding - Applying	7 – 8	Introduction to C Language ● Use Scratch program which is a Visual Block-based Programming Language to create a simple game	4
3	- Analysing - Evaluating - Creating	10 – 11	Control Structure ● Use Scratch for Arduino program with Electronic board (Arduino UNO) ● Control an LED circuit and small-sized motor	4

Activity 1: Introduction to Problem Solving

Entering week 3, the topic was Introduction to Problem Solving, which covered the procedures and steps in problem-solving. The students were expected to explain the algorithm with workflow and the thinking process. The Pseudocode was used to show the sequencing in the flowchart. Later on, the student-built up more understanding in the text-based programming language. The Flowgorithm, an application that creates programs using simple flowcharts, allowed the student to write and execute programs. It assisted the student in emphasizing on the algorithm rather than the syntax of a specific programming language. Figure 1 shows a screen capture of Flowgorithm. This activity expected students to review the meaning of symbols used in the flowcharts, and 3 control structures; namely, structure sequence, structure selection and structure repetition.

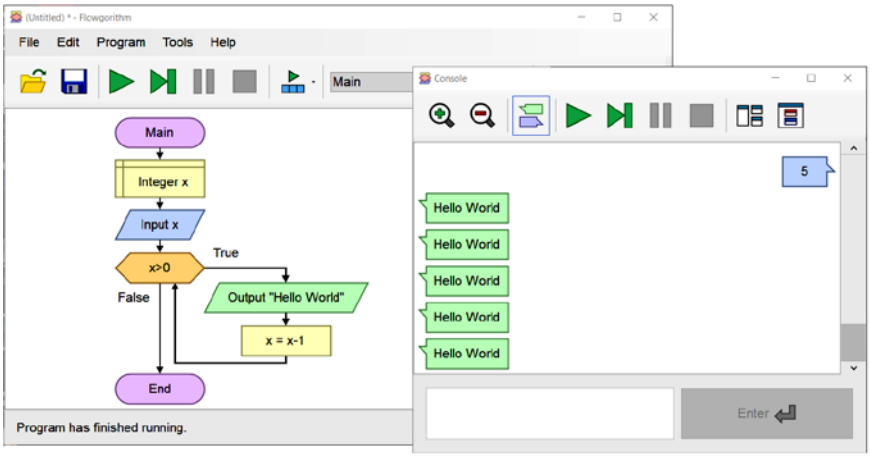

Figure 1. Screen capture of Flowgorithm program

Activity 2: Introduction to C Language

During week 7-8, the topic was structured programming languages, preparing the students to learn the text-based programming language. Once the students developed an understanding of programming logic, it is relatively easy for them to start learning one of the major programming languages. Thus, for the 2nd activity, a Scratch program was introduced to the students. The visual block-based programming language allows the student to program their own interactive stories, games, and animations. As a result, Scratch helps students engage more in class and show good signs of creative thinking, systematic thinking, engineering reasoning, and team collaboration. Figure 2 shows a screen capture of a Scratch program.

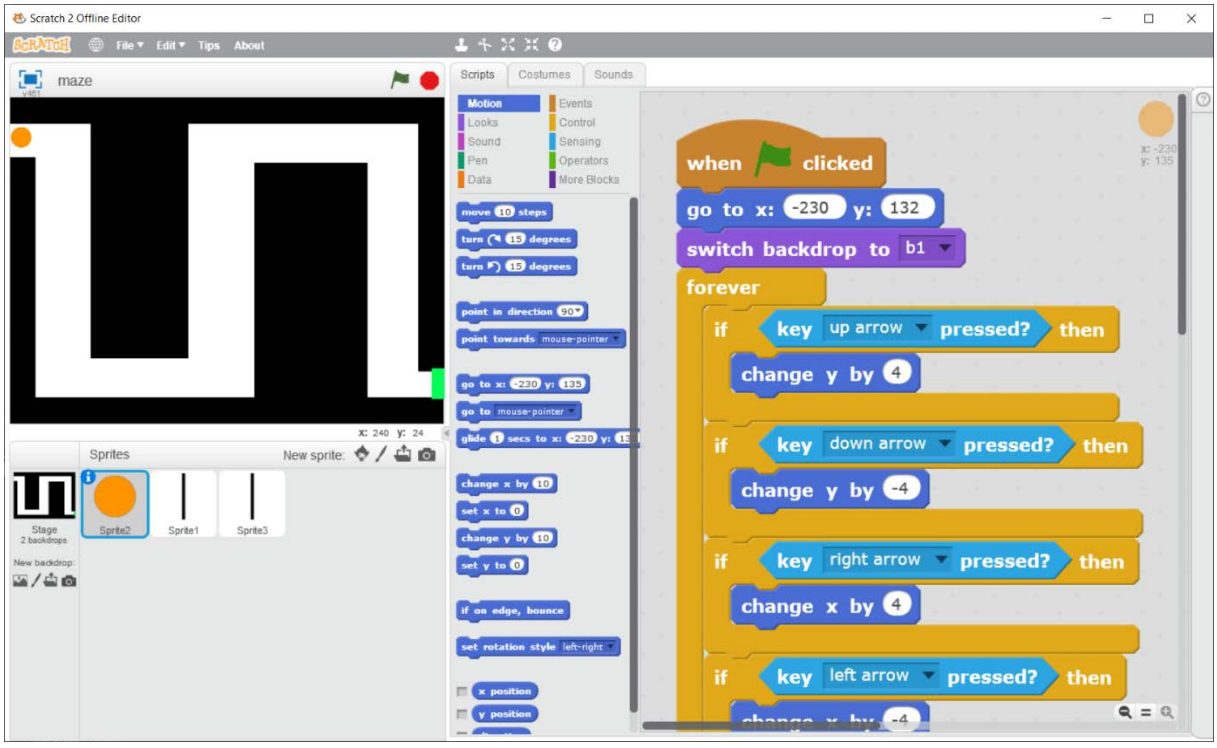

Figure 2. Screen capture of a Scratch program

Activity 3: Control Structure

For week 10-11, the students started using text-based programming languages. A majority of the students had difficulty understanding this content due to the increased level of complications and difficulties. This was the main cause of the students decreasing interest and motivation for coding. In order to overcome those challenges, Scratch program for Arduino board (S4A) was introduced to keep the students interested and motivated.

S4A is a Scratch modification that permits simple programming of the Arduino open-source hardware platform, containing a new set of blocks for managing sensors and actuators. The program itself can be connected to an Arduino microcontroller board which directly uploads control codes through the USB socket. With these features, the students are able to do tasks such as selecting blocks to turn on and off an LED light bulb and to rotate the servomotors. Figure 3 shows a hands-on practice of a student using the S4A in an active learning lesson.

Figure 3. Example of student in-class activity

RESULT

Statistical Test

The Class A (experimental group) and Class B (control group) students took a 50-multiple-choice questions (50 points) pre-test on week 2. The same questions are used for the post-test on week 11. The mean scores of both groups were compared and statistically tested by an independent sample t-test with a confidence level of 95%.

Table 4. T-test for Equality of Means Pre-test for Class A and Class B

	N	Mean	S.D.	Mean Difference	t	df	Sig 1 tailed
Class A	39	8.18	3.88	0.42	0.501	75	0.309
Class B	38	7.76	3.40				

From Table 4, the comparison between 2 groups from the pre-test results in week 2 shows that Class A average score was 8.18, and Class B averaged 7.76. The mean difference was 0.42. The sig (1-tailed) value of 0.309 was > 0.05. Therefore, we accepted the null hypothesis that there were no differences between Class A and Class B at the significant level of 0.05.

Table 5. T-test for Equality of Means Post-test for Class A and Class B

	N	Mean	S.D.	Mean Difference	t	df	Sig 1 tailed
Class A	39	23.36	7.01	4.37	3.004	75	0.002
Class B	38	18.63	6.79				

From Table 5, the comparison between 2 groups from the post-test results in week 11 showed that the mean score for Class A was 23.36 and 18.63 for Class B. The mean difference was 4.37. The sig (1-tailed) value of 0.002 was < 0.05. Therefore, the null hypothesis was rejected.

We can conclude that the mean score of Class A was higher than the mean score of Class B at the significant level of 0.05.

Student Feedback

At the end of the semester, the students gave feedback on their learning experience for the computer programming class. Table 6 shows a contrary of feedbacks between Class A and Class B students. The students in Class A that were offered active learning activities remained their motivation throughout the semester and achieved the learning outcome, in the process building a positive attitude towards the computer program. Meanwhile, Class B students showed distress and difficulty in grasping the concepts of computer programming.

Table 6. Feedbacks from the students after the semester ended

Class A (experimental group) Feedback	Class B (control group) Feedback
The activities help me understand with step-by-step explanation from the teacher.	*I didn't understand what you taught.*
The teacher did not rush when teaching. The good pace helps me who is a slow learner understand the subject.	*The examination was very difficult*
In the beginning, I didn't like this subject at all. Then, I understood and started to feel that it was actually fun.	*The teacher showed examples on the screen. I had no clue what programming is about.*

DISCUSSION AND CONCLUSION

The design and development of active learning activities were based on the linkage of topics, learning style, and learning outcome. The results of the study conclude that not only was the expected learning outcomes achieved, but the student's engagement and motivation were also maintained throughout the entire semester. According to Leong et al. (2016), the motivation that drives the students is directly affected and impacted by different settings: classroom characteristics, pedagogical approaches, physical environments, collaborative teams, and student autonomy. Students in the experiment group had experiences in a self-paced learning based classroom, hands-on pedagogical methods, visual and physical devices (Scratch and S4A) and an autonomous learning environment.

The research findings conclude that active learning activities can support the computational thinking process for the students. The students have achieved the expected outcomes including problem-solving and algorithm refining and reviewing, computational thinking, flowcharts writing, coding and computer programming. The experimental group students were satisfied with the course with positive attitudes and learning motivation towards computer programming. This is similar to Vega et al. (2013) findings, where the students' interests in polished and attracting activities resulted in an increase of the student's motivation. The visual block-based programming in the active learning sessions alongside hands-on practices using Flowgorithm, Scratch and S4A successfully supported the students in learning computer programming, with paralleling to the results from Gupta et al. (2012), Roscoe et al. (2014) and Tangney et al. (2010). Moreover, the level of student's satisfaction and motivation was pleasant, similarly to Siong and Thow (2017) findings that the learning-by-doing method can enhance the students' motivation.

The effort of supporting the 1st year non-programming engineering students learning computer programming was successful. The students had a positive attitude towards the course and proved that it is not extremely challenging and can be enjoyable. The course can be applied and extended to a larger scale, considering there are 10 faculty members who teach the subject. However, the teacher should be able to observe and assess the student's background knowledge, as well as their willingness and eagerness to learn new things. Future work will be the implementations of project-based approaches in the course. A programming contest environment can drive challenges in promoting motivation and self-directed learning within students.

REFERENCES

Anderson, L. W., Krathwohl, D. R. (2001). Bloom's Taxonomy Revised: Understanding the New Version of Bloom's Taxonomy, obtained on 2013 from https://thesecondprinciple.com/teaching-essentials/beyond-bloom-cognitive-taxonomy-revised/

Berglund, E., Persson, D. (2018). Turning Programming into a Relevant Topic for Nonprogramming Engineers, Proceedings of the 14th International CDIO Conference, Kanazawa, Japan: Kanazawa Institute of Technology, pp. 649-658.

Biggs, J. B., & Tang, C. (2011). Teaching for Quality Learning at University: What the Student Does. Maidenhead: Society for Research into Higher Education & Open University Press.

Gupta, N., Tejovanth, N., Murthy, P. (2012). , Learning by Creating: Interactive Programming for Indian High Schools, 2012 IEEE International Conference on Technology Enhanced Education (ICTEE), Kerala, India.

Hladik, S., Behjat, L., Nygren, A. (2017). Modified CDIO Framework for Elementary Teacher Training in Computational Thinking, Proceedings of the 13th International CDIO Conference. Calgary, Canada: University of Calgary, pp. 581-594

Leong, H., Shaun, A., Singh, M. (2016). Enhancing Students Self-Directed Learning and Motivation, Proceedings of the 12th International CDIO Conference, Turku, Finland: Turku University of Applied Sciences.

Meikleham, A., Hugo, R., Brennan, R. (2018). Fluid Mechanics Project-Based Learning Kits: An Analysis Of Implementation Results In A Blended Classroom. Proceedings of the 14th International CDIO Conference, Kanazawa, Japan: Kanazawa Institute of Technology, pp. 649-658.

Ortega-Sanchez, C. (2014). Curtin Robotics Club: Conceiving, Designing, Implementing and Operating Robots for Fun! Proceedings of the 10th International CDIO Conference, Barcelona, Catalonia, Spain: Universitat Politècnica de Catalunya.

Roscoe, J.F., Fearn, S., Posey, E., (2014). Teaching Computational Thinking by Playing Games and Building Robots, 2014 International Conference on Interactive Technologies and Games. Nottingham, Nottinghamshire, United Kingdom.

Shorn, S. (2018). Teaching Computer Programming Using Gamification, Proceedings of the 14th International CDIO Conference, Kanazawa, Japan: Kanazawa Institute of Technology, pp. 763-773.

Siong, G., Thow, Vivian. (2017). The Effect of Using "Learning-By-Doing" Approach on Students' Motivation in Learning Digital Electronics, Proceedings of the 13th International CDIO Conference, Calgary, Canada: University of Calgary, pp. 194-203.

Tangney, B., Oldham, E., Conneely, C., Barret, S., Lowler, J. (2010). Pedagogy and Processes for a Computer Programming Outreach Workshop - The Bridge to College Model, IEEE Transactions On Education, Vol. 53, No. 1, February 2010. pp. 53-60.

Tyler, R.W., Gagne, R.M., Scriven, M. (1967). Perspectives of Curriculum Evaluation. Chicago: Rand McNally.

Vega, M., Morales, E., Munoz, J. (2012). Innovating in Teaching And Learning at First Year Of Engineering, Proceedings of the 9th International CDIO Conference, Massachusetts Institute of

Technology and the Harvard School of Engineering and Applied Sciences, Cambridge, Massachusetts, USA.

Wing., J. (2006). Computational Thinking. Communications of the ACM, March 2006/Vol. 49, No. 3, 33-35.

Yongquian, C., Xiaojun, W., Chengbin, Q. (2018). Computer Programming Education for Primary School Students, Proceeding of the 13th International Conference on Computer Sciences & Education, Colombo, Sri Lanka, pp. 163-167.

BIOGRAPHICAL INFORMATION

Natha Kuptasthien is currently as assistant to president for International Relations and an associate professor at the industrial engineering department, faculty of engineering, RMUTT. She led a full CDIO implementation at RMUTT since 2013. She has conducted a number of CDIO introductory workshops for engineering and non-engineering programs, which expanded the CDIO network to 8 RMUTs and universities in Asia. Natha graduated with a Bachelor of Engineering in Industrial Engineering from Chulalongkorn University, Master of Science and PhD in Engineering Management from University of Missouri-Rolla, USA.

Deachrut Jaithavil is a lecturer at the Department of Computer Engineering, Faculty of Engineering, RMUTT. His pedagogical interest is design and development of computer programming courses including digital electronics and microcomputers.

Corresponding author

Mr. Deachrut Jaithavil
Rajamangala University of Technology Thanyaburi (RMUTT)
39 Village No. 1, Rangsit-nakornnayok Road, Klong 6, Thanyaburi, Pathumthani, Thailand 12110
deachrut.j@en.rmutt.ac.th