Using Variation Theory as a Guiding Principle in an OOP Assisted Syntax Correction Learning System

https://doi.org/10.3991/ijet.v15i14.14191

Ming Che Lee
Ming Chuan University, Tauyuan, Taiwan

Jia Wei Chang(✉)
National Taichung University of Science and Technology, Taichung, Taiwan
jiaweichang.gary@gmail.com

Tzone I Wang, Zi Feng Huang
National Cheng Kung University, Tainan, Taiwan

Abstract—Object-oriented programming skill is important for the software professionals. It has become a mandatory course in information science and computer engineering departments of universities. However, it is hard for novice learners to understand the syntax and semantics of the language while learning object-oriented programming, and that makes them feel frustrated. The purpose of this study is to build an object-oriented programming assistant system that gives syntax error feedback based the variation theory. We established the syntax correction module on the basis of the Virtual Teaching Assistant (VTA). While compiling codes, the system will display syntax errors, if any, with feedbacks that are designed according to the variation theory in different levels (the generation, contrast, separation, and fusion levels) to help them correcting the errors. The experiment design of this study splits the participants, who are university freshmen, into two groups by the S-type method based on the result of a mid-term test. The learning performances and questionnaires were used for surveying, followed by in-depth interviews, to evaluate the feasibility of the proposed assistant system. The findings indicate that the learners in the experimental group achieved better learning outcomes than their counterparts in the control group. This can also prove that the strategy of using the variation theory in implementing feedback for object-oriented programming is effective.

Keywords—Variation Theory; Object-Oriented Programming; Virtual Teaching Assistant

1 Introduction

Object-Oriented Programming (OOP) is a programming paradigm. An object is a collection of classes. It uses objects as the basic unit of a program and encapsulates programs and data in it to improve software reusability, flexibility, and scalability. Object-oriented programming can be thought of as the idea of including a variety of independent and mutually invoking objects in a program. This is the opposite of traditional thinking: traditional programming advocates treating programs as a collection of functions, or a series of instructions given to the computer. The design of OOP is based on the objects to be processed by a program and emphasizes that the cooperation between various independent components is required to solve problems [1]. For novice learners, understanding the concepts of OOP can be challenging. Previous studies on OOP learning have revealed that when beginning to learn programming, students often do not fully understand the concepts of programming, are unfamiliar with programming languages (particularly programming syntax and statements), experience misconceptions about different concepts, and thus feel discouraged [2-6]. Some researchers have indicated that students often do not understand some concepts such as recursive [7], function [8] or other basic OOP conceptions such as class, object, interaction, inheritance, and polymorphism [9-13].

Originated from phenomenography [14], variation theory states that people often experience changes before engaging in learning; in other words, people learn new concepts by distinguishing between phenomena or concepts and by comparing their current and past experiences. Based on this theory, four methods can be applied: generation, contrast, separation, and fusion. Many researchers have claimed that variation theory helps students clarify their misconceptions or fuzzy concepts. Variation theory has been shown to be an effective and feasible teaching strategy [14-17].

To help the OOP beginners who do not fully understand programming syntax, semantics, and statements and consequently become frustrated, this study used variation theory to establish an OOP-based syntax correction system. Accordingly, students can use this system to practice writing C++ object-oriented programs and compiling source codes. In addition, when students must debug their programs, this syntax correction system can help them correct syntax errors in accordance with variation theory.

2 Literature Review

Variation theory is an instructional theory that primarily intends to identify particular ways of understanding things [18]. According to variation theory, experiencing changes facilitates initiating the process of beginning [14][19]. In other words, people learn new concepts by distinguishing between different phenomena or concepts. Variation theory has been applied to several different topics, such as economic supply and demand for primary and high school students [20], chemical engineering [21-22], classical physics for freshman physics students [21][23], and fluid mechanics for engineering students [24].

Variation theory was derived from phenomenography [14]. The purpose of phenomenography is to examine how people perceive an identical phenomenon differently. To perceive or experience a phenomenon, a person must undergo three processes: discernment, variation, and simultaneity [25-27]. Discernment is the ability of a person to distinguish some characteristics of an object from other characteristics of the object. Variation means that after experiencing changes in an entity, a person can detect critical features of the entity. Simultaneity means that a person simultaneously experiences changes in several critical characteristics of an event or entity.

Such perception processes can contribute towards understanding how students learn. According to variation theory, differences and changes initiate cognition. For students to learn new knowledge, they must pay attention to differences between dissimilar events or objects. Stepans [28] proposed a conceptual change model, which states that students should be aware of the differences between their own concepts and other concepts. In other words, when certain dimensions of an entity remain constant while its other dimensions change, these changes in the dimensions will be detected. A complete variation theory must comprise four variation models: generation, contrast, separation, and fusion [29-30], as listed in ascending order of difficulty.

2.1 Generation

To understand the universality of an entity, people must experience the entity in various situations. To learn a new concept, students are required to detect the common features of several examples related to the concept.

2.2 Contrast

To understand an entity clearly, people must contrast the entity with other entities. Concretely speaking, if a person wishes to understand X, the person must also understand non-X events.

2.3 Separation

To discern some characteristics of an entity, people must experience the changes of the entity in some respects while other aspects of the entity remain unchanged. Accordingly, students should consider various solutions to a problem or various explanation for a phenomenon.

2.4 Fusion

When several factors that influence an entity are considered, changes in several critical characteristics of the entity must often be performed and detected. For example, to teach elementary school students a new word, a teacher must associate the shape of the word with the sound and meaning of the word and simultaneously manipulate the shape, sound, and meaning of the word.

Variation theory can help students clarify their misconceptions or uncertain concepts and can be used to develop an effective and feasible teaching strategy. Previously, variation theory has been applied in conventional programming teaching. Suhonen, et al. [31] applied variation theory to teach heterogeneous students and to highlight the varying characteristics and conceptual implications of programming; the results indicated that variation theory was an effective method for presenting the critical characteristics of programming and for helping students learn core programming concepts. Thuné, and Eckerdal [32] applied variation theory to identify the programming learning pattern of university students and to help them solve their problems with learning programming. Thota and Whitfield [33] applied variation theory and constructivism to help students learn OOP effectively and resulted in desirable learning outcomes.

Eckerdal and Thuné [34] used variation theory to assist first-year university students in distinguishing the concepts of object and class in OOP. The said scholars indicated that an “object” in OOP can be considered as an active component of a program or a phenomenon module, whereas a “class” can be considered as a template for a set of source codes, an object attribute, or a phenomenon. Moreover, they used variation theory based on phenomenography to teach students how object differed from class. In this manner, students effectively learned how to distinguish between the two entities. According to aforementioned studies, variation theory is an effective and feasible method for teaching traditional programming design and OOP.

3 Methods

Programming beginners often do not fully understand programming concepts, particularly the uses of syntax or statements, during OOP learning and thus become frustrated [35-36]. In this study, we used variation theory to establish an Internet OOP-learning environment to help students correctly understand programming syntax and concepts during OOP design.

In accordance with our strategy, students were requested to write an object-oriented program. Figure 1 shows the procedure of preliminary syntax correction that was employed in this study. If compilation errors occurred, the OOP syntax correction system would guide students to correct their errors. When the students modified and compiled their source codes, the syntax correction system would provide adequate feedback to the students in accordance with variation theory.

We established the syntax correction module on the basis of the Virtual Teaching Assistant (VTA) proposed by Chou et al. [37] to help students debug programs. The empirical results verified the feasibility of the VTA. In the present study, we simplified the VTA, integrated it with variation theory, and used the syntax errors database and the syntax correction database to teach programming, as shown in Figure 2.

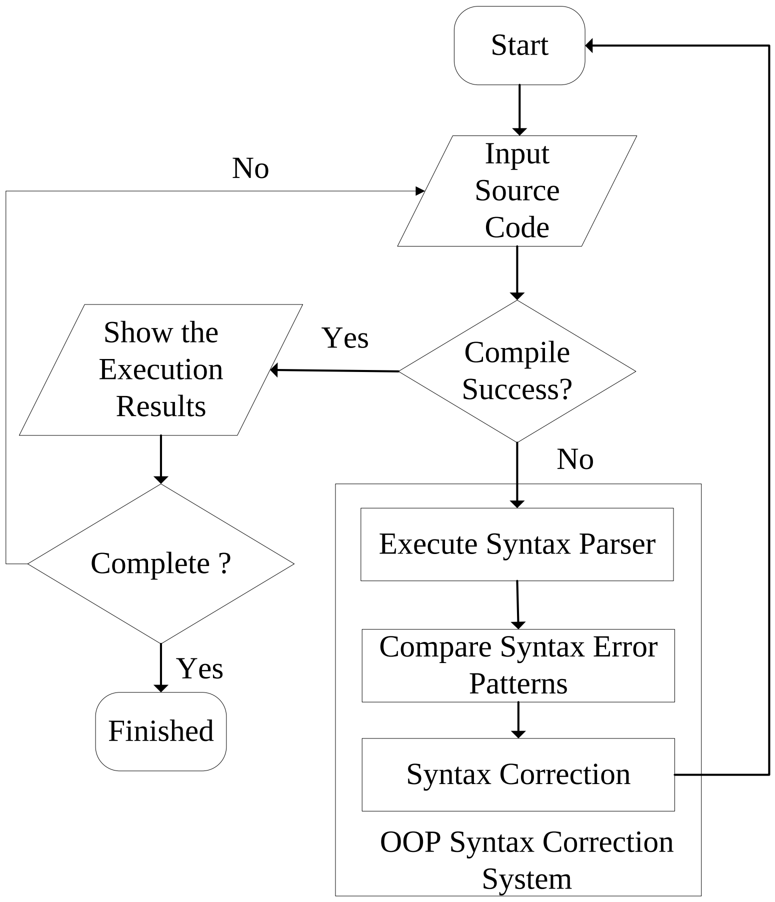


Fig. 1. The Variation Theory based Object-Oriented Programming Syntax Correction System

As illustrated, hint 1 points out the error locations and types according to the base compiler. Hint 2 starts the variation theory correction process. When students repeatedly encounter the same program error, the system will provide the easy-to-challenging feedbacks step by step. Figure 3 illustrates a programming example of access errors in a private area. The errors occurred because an object mistakenly called a member function in the private area. In C++, if data members or member functions of a particular class are confined to a private area, then these members can only be accessed through member functions of the same class.

On the basis of this example, teaching materials related to syntax correction for the four variation models in variation theory are described as follows:

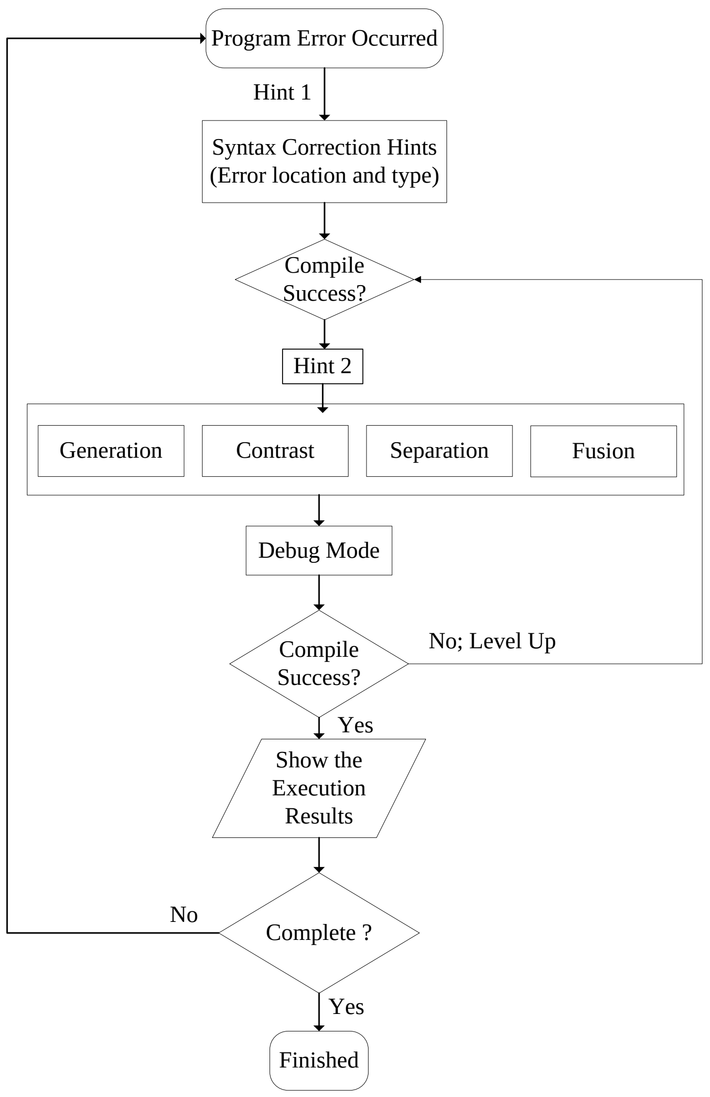


Fig. 2. Correction Process of the proposed framework

Level 1—The generation model

In the syntax correction system, the generation model enumerates two to three examples that are similar to the aforementioned example related to access errors in a private area. Accordingly, students can understand their programming errors by observing these examples.

Level 2—The contrast model

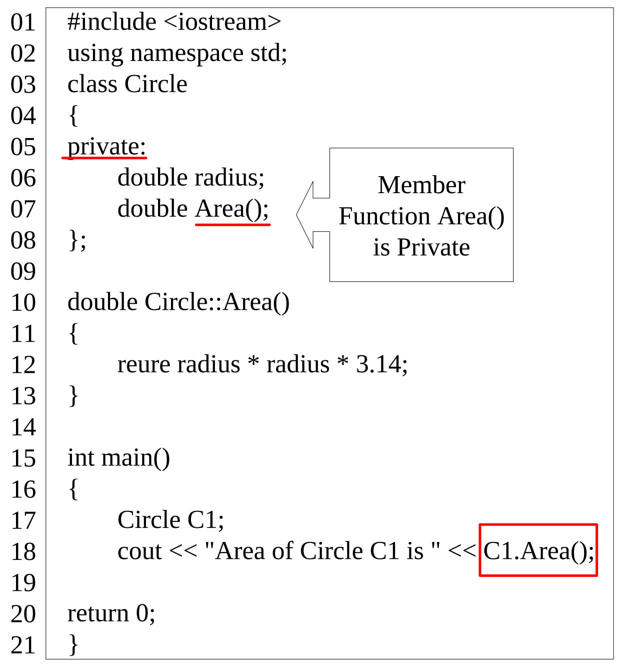
Under the contrast model, if access errors in a private area are being shown as an example, then the correct answer being taught should be that data members should be confined to a public area. According to variation theory, to teach students what a public area is, a teacher should demonstrate to them what a nonpublic (private) area is. Students should learn similarities and differences between members in a private area and those in a public area to understand why related programming errors occur.

Level 3—The separation model

To use access errors in a private area as an example, the syntax correction system provides two to three relevant correct programs to students. Accordingly, the students can experience the changes in the erroneous programs (changes from incorrect to correct programs), compare them with the correct programs, and learn how to write correct programs.

Level 4—The fusion model

To use access errors in a private area as an example, the syntax correction system provides a program that contains both the correct and incorrect source codes. After students examine the aforementioned three types of teaching material and can identify the characteristics of an entity, they further integrate all the characteristics of the entity under the same scenario. Accordingly, the students can fully and correctly understand programming syntax.



```
#include <iostream>
using namespace std;
class Circle
{
private:
        double radius;
        double Area();
};

double Circle::Area()
{
        reure radius * radius * 3.14;
}

int main()
{
        Circle C1;
        cout << "Area of Circle C1 is " << C1.Area();

return 0;
}
```

Fig. 3. A programming example of access errors in a private area

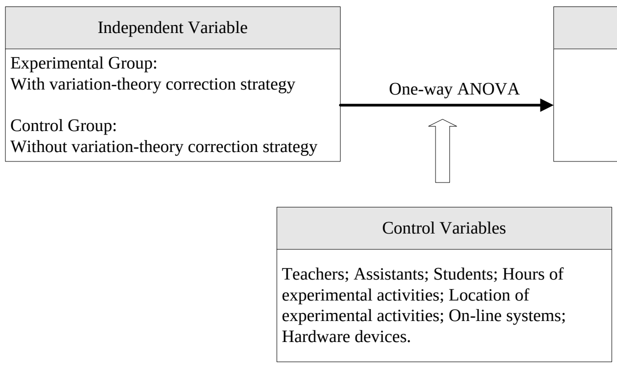

Independent Variable
Experimental Group:
With variation-theory correction strategy
Control Group:
Without variation-theory correction strategy
One-way ANOVA
Dependent Variable
Learning Performance
Control Variables
Teachers; Assistants; Students; Hours of experimental activities; Location of experimental activities; On-line systems; Hardware devices.

Fig. 4. The experimental framework

4 Experimental Design

4.1 Participants

A total of 90 students that attended the compulsory programming course provided by the department of engineering science at National Cheng Kung University were recruited in this study (45 students in the experimental group and 45 students in the control group). The students who withdrew from the course or were absent from experimental activities were exclude from this study. Therefore, for the first experiment (Experiment 1: Object and Class), the control group comprised 42 students and the experimental group comprised 40 students. For the second experiment (Experiment 2: Inheritance), the control group comprised 41 students and the experimental group comprised 40 students.

4.2 Experimental design

To examine whether the "variation-theory and OOP syntax-correction integration system" developed in the present study effectively helped students learn OOP, experimental activities, interviews, and a questionnaire survey were conducted after the various modules for this system were constructed.

In this study, we applied an S-grouping method and assigned students to the experimental or control group in accordance with their midterm examination results. Accordingly, the two groups of students had the same level of prior knowledge about programming. Figure 4 illustrates the experimental framework of this study. When programming errors occurred, the syntax correction module for the control group presented Hint 1 to the students and informed them of error locations and types. If the students still could not debug the incorrect syntax, then the syntax correction module presented Hint 2 to the students and provided the students with the base compiler information (Microsoft Visual C++) to help them modify the incorrect syntax.

4.3 Experimental procedures

Before conducting formal experiments, we performed preparation tasks such as contacting the participating students, collecting the student information, and grouping the students. In this study, we conducted two experiments. After explaining the experimental procedures to the students, we asked them to complete a test on a computer. During Experiment 2, the students completed a questionnaire after receiving a test. After the two experiments were completed, some students from the experimental and control groups were interviewed according to the phenomena observed in the experiments.

4.4 Research instruments

The research instruments in this study comprised a prior knowledge test, an OOP syntax correction system, a syntax correction test, and a questionnaire regarding the use of the syntax correction system that incorporated variation theory:

The prior knowledge test: In this study, the first midterm test of the programming course provided by the department of engineering science at National Cheng Kung University was used as the prior knowledge test. The scope of this test involved control procedures, functions, arrays, indexes, and strings. The test results were used to assess the prior knowledge of the participants in this study.

The OOP syntax correction system: The OOP syntax correction system provided an Internet learning environment, enabling students to use this system to write a C++ OOP program. Additionally, when programming errors occurred, the system presented hints and the syntax correction modules helped the students debug their programs.

The syntax correction test: The syntax correction test was used to examine whether the OOP syntax correction system effectively enhanced students' learning performance. Two experiments were conducted in this study. The scope of the test in Experiment 1 was object and class; the scope of the test in Experiment 2 was inheritance. In accordance with the scopes of the tests, several questions about programming were presented in the tests. The computer tests used in the experiments served as the syntax correction tests. The test questions and scoring methods were fabricated by teachers and experts who were experienced in OOP teaching; hence, the test questions and scoring methods possessed adequate expert validity. The test results were used to assess the students' learning performance.

5 Experimental Results

5.1 Learning performance

A total of 90 students were recruited in the experimental activities. Some students withdrew or were absent from the experimental activities; thus, the valid sample size was 82 for Experiment 1 and 81 for Experiment 2. In this study, a one-way ANOVA was performed to test whether a significant difference in variance existed between the experimental and control groups. The significance level was set as .05.

Prior to the formal experiments, a Levene's homogeneity test of variances was performed to examine the homogeneity of variances between the experimental and control groups. The results showed that no significant difference in variances existed between the two groups (p=.891 and p=.976). Therefore, the homogeneity assumption was confirmed.

Table 1. One-way ANOVA; Learning Performance

		Sum of Squares	df	Mean Squares	F	Sig.
E_1	Between Groups	4842.4	1	4842.4	4.255	0.042*
	Within Groups	91054.0	80	1138.2		
	Total	95896.5	81			
E_2	Between Groups	4240.4	1	4240.4	4.165	0.045*
	Within Groups	80420.9	79	80420.9		
	Total	84661.3	80			

*P< 0. 05

Table 2. Mean and standard deviation of the Questionnaire

	No.	Effective Samples	Mean	Standard Deviation
Generation model	#1	40	3.58	.675
	#2	40	3.40	.709
	#3	40	3.45	.749
Contrast model	#4	40	4.03	.480
	#5	40	3.75	.588
Separation model	#6	40	3.85	.662
	#7	40	3.65	.802
Fusion model	#8	40	3.70	.648
	#9	40	3.63	.740
Overall experience	#10	40	3.65	.622
	#11	40	3.50	.716
	#12	40	3.78	.660

As shown in Table 1, a one-way ANOVA analysis was performed to assess whether a significant difference in learning performance existed between the experimental and control groups. For Experiments 1 and 2, the results attained statistical significance (F (1,80) = 4.255, p < 0.042 for Experiment 1; F (1,79) = 4.165, p < 0.045 for Experiment 2). Therefore, the learning performance of the students who used the OOP syntax correction system that incorporated variation theory was superior to that of the students who used the OOP syntax correction system that did not incorporate variation theory.

Table 3. Subjects' Assessment for the Generation Model

Item	SA.	A.	N.	D.	SD.	Avg.
The feedback of the system provides 2-3 "*programming examples with the same errors*" as a reference can help me to identify the errors and locations.	2, 5%	21, 52.5%	15, 37.5%	2, 5%	0, 0%	3.575
The feedback of the system provides 2-3 "*programming examples with the same errors*" as a reference can help me to clarify the programming syntax.	1, 2.5%	18, 45%	17, 42.5%	4, 10%	0, 0%	3.4
I can identify the same error type while encountering the same syntax error in the next time.	2, 5%	18, 45%	16, 40%	4, 10%	0, 0%	3.45

5.2 Questionnaire survey

The questionnaire survey was administered to the experimental group to examine their opinions and suggestions about the use of the variation-theory and syntax-correction system and how the four models of variation theory helped them debug their programs. Table 2 displays the survey results. Most of the students in the experimental group agreed for the three items regarding overall experience (the mean scores were 3.65, 3.50, and 3.78). In addition, most students agreed for the corrective feedback provided by the four models of variation theory. Particularly, the largest number of students considered that the contrast model was most helpful (the mean scores were 4.03 and 3.75), followed by the separation model (the mean values were 3.85 and 3.65), the fusion model (the mean scores were 3.70 and 3.63), and the generation model (the mean scores were 3.58, 3.40, and 3.45). Further details about the survey results related to the four models and overall experience are provided as follows.

The generation model: As shown in Table 3, for Item 1, 57.5% of the students in the experimental group considered the generation model helpful for identifying programming errors; 42.5% of the students had no comments or showed disagreement. For Item 2, 47.5% of the students considered the generation model helpful for clarifying programming syntax concepts; 52.5% of the students had no comments or showed disagreement. For Item 3, 50% of the students considered the generation model helpful for identifying identical programming syntax errors in other programs; 50% of the students had no comments or showed disagreement.

The contrast model: As shown in Table 4, for Item 4, 90% of the students in the experimental group considered the contrast model helpful for understanding why programming errors occurred; 10% of the students had no comments. For Item 5, 72.5% of the students agreed that the contrast model was helpful for clarifying programming syntax concepts; 27.5% of the students had no comments or disagreed.

The separation model: As shown in Table 5, for Item 6, 70% of the students in the experimental group agreed that the comparison between correct and incorrect programs helped them understand correct programming syntax; 30% of the students had no comments or disagreed. For Item 7, 65% of the students considered the comparison between correct and incorrect programs helpful for analysing programming errors; 35% of the students had no comments or showed disagreement.

The fusion model: As shown in Table 6, for Item 8, 65% of the students in the experimental group agreed that the fusion model helped them fully understand incorrect programming syntax and conduct correction; 35% of the students had no comments or disagreed. For Item 9, 57% of the students agreed that the fusion model helped them fully understand correct programming syntax; 43% of the students had no comments or disagreed.

Overall experience: Table 7 shows how the experimental group perceived the variation-theory and syntax-correction system. Regarding Item 10, 27 students (67.5%) agreed that the feedback information provided by the system helped them clarify fuzzy syntax concepts and correctly understand programming syntax. For Item 11, 23 students (57.5%) agreed that how the system gradually provides feedback helped them profoundly understand programming syntax. For Item 12, 30 students (75%) agreed that the feedback information provided by the system helped them to correct programming syntax errors.

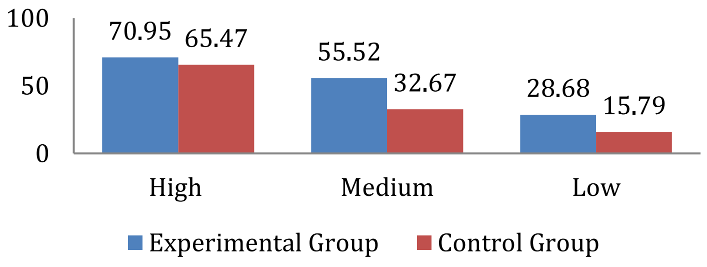


Fig. 5. The mean score of the two tests completed by students in the experimental and control group

6 Discussion

During the experimental processes, we observed numerous phenomena. Interviews were administered to some students who participated in the experiments regarding these phenomena. This section also addresses how the OOP syntax correction system influenced the students and how the variation-theory corrective feedback influenced the students.

6.1 The influence of the OOP syntax correction system on the students

First, in accordance with the prior knowledge test results (the first midterm test results), the students were divided into three groups: the high, medium, and low achievement groups. The high achievement group comprised the students in the first score quartile; the medium achievement group comprised the students in the second and third score quartiles; the low achievement group comprised the students in the fourth score quartile. Figure 5 displays the mean score of the two tests completed by each student in the experimental and control group during Experiments 1 and 2. As shown in Fig. 5, for the high, medium, and low achievement groups, the average scores of the experimental group were higher than those of the control group by 5.48, 22.85, and 12.89 points, respectively.

The results indicated that the OOP syntax correction system that incorporated variation theory was most useful to the medium achievement group, followed by the low achievement group, and was slightly helpful to the high achievement group. Subsequently, interviews were administered to the students to examine their opinions about the Internet learning system. The students indicated that the provided feedback information about error positions and programming error types helped them effectively correct their programming errors when they already possessed a basic understanding of the errors. For example, a student from the control group said "I feel that feedback information about error positions and error types are useful for debugging programs because if most errors are of the same type, we can rapidly use the same method to correct these errors."

6.2 The influence of the variation-theory corrective feedback on the students

In this section, we explored how the generation, contrast, separation, and fusion models influenced the students. Most students reported that the variation-theory corrective feedback was useful for programming. The survey response results and interview content are further discussed as follows:

The influence of the generation model: The generation model enumerated numerous examples related to programming errors to help the students identify the errors in their programs. Table 3 displays the related survey results. Approximately half of the students (57.5%, 47.5%, and 50%) agreed that the feedback information provided by the generation model was useful; the remaining students (42.5%, 52.5%, and 50%) had no comments or disagreed. The reason may be that the feedback information provided by the generation model assisted the students in correcting simple programming errors but was not useful for solving complex problems. For example, a student from the experimental group said "I think that feedback information can help solve simple problems such as identifying a missing semicolon."

The influence of the contrast model: The contrast model informed the students of programming errors and how to correct the errors; therefore, they could compare incorrect programs and correct programs to clarify their programming syntax concepts. Table 4 shows the related survey results. Most students agreed that the feedback information provided by the contrast model was useful. Particularly, 90% of the students agreed the item that "the feedback information provided by the contrast model helped them understand why programming errors had occurred;" of all the items, the highest proportion of the students agreed with this item. In addition, 72.5% of the students reported that the feedback information provided by the contrast model helped them clarify programming syntax concepts. A student from the experimental group said "I think that the online system was useful; it provided feedback to help me correct errors; accordingly, I could clearly know where programming errors had occurred."

The influence of the separation model: The separation model provided examples of correct programs to help the students resolve their programming errors. In other words, the students compared the correct programs provided by the separation model and their own incorrect programs to understand how to write correct programming syntax. According to the results of this study, most students (70% and 65%) agreed that the feedback information provided by the separation model was useful and that it helped them understand correct programming syntax and analyse programming errors. Some students indicated that the feedback information provided by the separation model was extremely useful for helping them solve complex programming problems. For example, a student from the experimental group said "by comparing the correct examples with my own programs, I solved complex problems; I encountered a problem where a void function returned a value; I could not solve this problem only by viewing incorrect examples."

The influence of the fusion model: The fusion model provided feedback information containing both correct and incorrect examples. By highlighting the characteristics of the correct examples and the incorrect examples in the same situation, the fusion model helped students fully understand programming syntax. According to the survey results, 65% of the students reported that the feedback information provided by the fusion model helped them fully understand programming syntax errors and conduct correction to the errors; 57.5% of the students reported that the fusion model helped them analyse programming errors. The students indicated that the feedback information provided by the fusion model was useful. A student from the experimental group said "using an example to show correct and incorrect source codes was useful to me." Another student from the experimental group said "simultaneously presenting incorrect and correct examples helped me understand my mistakes."

7 Conclusion

This study explored the effectiveness of the proposed variation-theory and OOP-syntax-correction integration system. The results confirmed that regarding students' learning performance, the syntax correction system that incorporated variation theory was superior to the syntax correction system that did not incorporate variation theory. Therefore, the application of variation theory to OOP-syntax-correction learning helped the student participants write OOP programs and solve programming problems related to programming syntax, semantics, and statements.

According to the survey results and interview content, how the feedback information provided at various levels helped the students is described as follows:

1. If students understand how certain errors occur and how the errors can be corrected, then error positions and types are useful feedback information for helping the students correct their programming errors rapidly.
2. The feedback information about syntax correction provided by the generation model can help students solve simple programming problems.

3. The feedback information about syntax correction provided by the separation model can help students solve complex programming problems and correct complex programming errors.
4. The contrast model provides the most useful feedback information about how to correct programming errors.
5. The fusion model provides an example containing both incorrect and correct source codes, which can help students correct their programming errors.

Because of limited resources, this study only explored how variation theory could be applied in OOP syntax correction learning. Subsequent studies are suggested to explore how the application of variation theory to syntax correction learning influences and helps students at low, medium, or high achievement levels, and how the relevant applications assist students of heterogeneous competency levels.

8 References

[1] Martin, J., & Odell, J. J. (1994). Object-oriented methods. Prentice hall PTR.

[2] Anderson, J. R., Pirolli, P., & Farrell, R. (1988). Learning to program recursive functions. *The nature of expertise*, 153-184.

[3] Qian, Y., & Lehman, J. (2017). Students' misconceptions and other difficulties in introductory programming: A literature review. *ACM Transactions on Computing Education* (*TOCE*), 18(1), 1-24. https://doi.org/10.1145/3077618

[4] Fatourou, E., Zygouris, N. C., Loukopoulos, T., & Stamoulis, G. I. (2018). Teaching Concurrent Programming Concepts Using Scratch in Primary School: Methodology and Evaluation. International Journal of Engineering Pedagogy (iJEP), 8(4), 89-105. https://doi.org/10.3991/ijep.v8i4.8216

[5] Sunday, K., Ocheja, P., Hussain, S., Oyelere, S. S., Samson, B. O., & Agbo, F. J. (2020). Analyzing Student Performance in Programming Education Using Classification Techniques. International Journal of Emerging Technologies in Learning (iJET), 15(02), 127-144. https://doi.org/10.3991/ijet.v15i02.11527

[6] Zhao, Y., Guo, L., Liu, H., & Zheng, W. (2019). Design of Programming Experiment Course Platform Based on MOOCs. International Journal of Emerging Technologies in Learning (iJET), 14(10), 208-216. https://doi.org/10.3991/ijet.v14i10.10330

[7] Anderson, J. R., Pirolli, P., & Farrell, R. (1988). Learning to program recursive functions. *The nature of expertise*, 153-184.

[8] Xinogalos, S., Satratzemi, M., & Dagdilelis, V. (2006). An introduction to object-oriented programming with a didactic microworld: objectKarel. *Computers and Education*, *47*(2), 148-171. https://doi.org/10.1016/j.compedu.2004.09.005

[9] Abuaiadah, D., Burrell, C., Bosu, M., Joyce, S., & Hajmoosaei, A. (2019). Assessing Learning Outcomes of Course Descriptors Containing Object Oriented Programming Concepts. New Zealand Journal of Educational Studies, 54(2), 345-356. https://doi.org/10.1007/s40841-019-00139-y

[10] Eckerdal, A., & Thuné, M. (2005, June). Novice Java programmers' conceptions of object and class, and variation theory. In *ACM SIGCSE Bulletin* (Vol. 37, No. 3, pp. 89-93). ACM. https://doi.org/10.1145/1151954.1067473

[11] Kelter, R., Kramer, M., & Brinda, T. (2018, November). Statistical Frequency-Analysis of Misconceptions in Object-Oriented-Programming: Regularized PCR Models for Frequen-

cy Analysis across OOP Concepts and related Factors. In Proceedings of the 18th Koli Calling International Conference on Computing Education Research (pp. 1-10). https://doi.org/10.1145/3279720.3279727
[12] Schulte, C., & Bennedsen, J. (2006, September). What do teachers teach in introductory programming? In*Proceedings of the second international workshop on Computing education research* (pp. 17-28). ACM. https://doi.org/10.1145/1151588.1151593
[13] Teif, M., & Hazzan, O. (2006, June). Partonomy and taxonomy in object-oriented thinking: junior high school students' perceptions of object-oriented basic concepts. *ACM SIGCSE Bulletin* (Vol. 38, No. 4, pp. 55-60). ACM. https://doi.org/10.1145/1189136.1189170
[14] Marton, F. (1981). Phenomenography—describing conceptions of the world around us. *Instructional science*, *10*(2), 177-200. https://doi.org/10.1007/bf00132516
[15] Chiu, T. K., & Churchill, D. (2015). Exploring the characteristics of an optimal design of digital materials for concept learning in mathematics: Multimedia learning and variation theory. *Computers and Education*, *82*, 280-291. https://doi.org/10.1016/j.compedu.2014.12.001
[16] Kullberg, A., Runesson, U., Marton, F., Vikström, A., Nilsson, P., Mårtensson, P., & Häggström, J. (2016). Teaching one thing at a time or several things together? –teachers changing their way of handling the object of learning by being engaged in a theory-based professional learning community in mathematics and science. *Teachers and Teaching*, 1-15. https://doi.org/10.1080/13540602.2016.1158957
[17] Pang, M. F., Linder, C., & Fraser, D. (2006). Beyond lesson studies and design experiments–using theoretical tools in practice and finding out how they work. *International Review of Economics Education*, *5*(1), 28-45. https://doi.org/10.1016/s1477-3880(15)30126-2
[18] Vikström, A. (2008). What is intended, what is realized, and what is learned? Teaching and learning biology in the primary school classroom. *Journal of Science Teacher Education*, *19*(3), 211-233. https://doi.org/10.1007/s10972-008-9090-y
[19] Marton, F., & Runesson, U. (2004). The space of learning. In F. Marton., and ABM Tsui (Red.), *Classroom discourse and the space of learning* (pp. 3-40), Mahwah, NJ: Erlbaum.
[20] Pang, M. F., Linder, C., & Fraser, D. (2006). Beyond lesson studies and design experiments–using theoretical tools in practice and finding out how they work. *International Review of Economics Education*, *5*(1), 28-45. https://doi.org/10.1016/s1477-3880(15)30126-2
[21] Fraser, D., Allison, S., Coombes, H., Case, J., & Linder, C. (2006). Using variation to enhance learning in engineering. *International Journal of Engineering Education*, *22*(1), 102.
[22] Fraser, D., & Linder, C. (2009). Teaching in higher education through the use of variation: Examples from distillation, physics and process dynamics. *European Journal of Engineering Education*, *34*(4), 369-381. https://doi.org/10.1080/03043790902989507
[23] Linder, C., & Fraser, D. (2006). Using a variation approach to enhance physics learning in a college classroom. *The Physics Teacher*, *44*(9), 589-592. https://doi.org/10.1119/1.2396777
[24] Fraser, D. M., Pillay, R., Tjatindi, L., & Case, J. M. (2007). Enhancing the learning of fluid mechanics using computer simulations. *Journal of Engineering Education*, *96*(4), 381. https://doi.org/10.1002/j.2168-9830.2007.tb00946.x
[25] Marton, F., & Trigwell, K. (2000). Variatio est mater studiorum. *Higher Education Research and Development*, *19*(3), 381-395. https://doi.org/10.1080/07294360020021455
[26] Marton, F., & Runesson, U. (2004). The space of learning. In F. Marton., and ABM Tsui (Red.), *Classroom discourse and the space of learning* (pp. 3-40), Mahwah, NJ: Erlbaum.

[27] Pang, M. F. (2003). Two faces of variation: On continuity in the phenomenographic movement. *Scandinavian journal of educational research*,*47*(2), 145-156. https://doi.org/10.1080/00313830308612
[28] Stepans, J. (1996). Targeting students' science misconceptions: Physical science concepts using the conceptual change model. Idea Factory.
[29] Cheng, M. Y., & Ho, C. M. (2009). A study on applying the variation theory to Chinese communicative writing. *Asian Social Science*, *4*(10), 14. https://doi.org/10.5539/ass.v4n10p14
[30] Leung, A. (2008). Dragging in a dynamic geometry environment through the lens of variation. *International Journal of Computers for Mathematical Learning*,*13*(2), 135-157. https://doi.org/10.1007/s10758-008-9130-x
[31] Suhonen, J., Davies, J., & Thompson, E. (2007, November). Applications of variation theory in computing education. In*Proceedings of the Seventh Baltic Sea Conference on Computing Education Research-Volume 88* (pp. 217-220). Australian Computer Society, Inc.
[32] Thuné, M., & Eckerdal, A. (2009). Variation theory applied to students' conceptions of computer programming.*European Journal of Engineering Education*, *34*(4), 339-347. https://doi.org/10.1080/03043790902989374
[33] Thota, N., & Whitfield, R. (2010). Holistic approach to learning and teaching introductory object-oriented programming. *Computer Science Education*, *20*(2), 103-127. https://doi.org/10.1080/08993408.2010.486260
[34] Eckerdal, A., & Thuné, M. (2005, June). Novice Java programmers' conceptions of object and class, and variation theory. In*ACM SIGCSE Bulletin* (Vol. 37, No. 3, pp. 89-93). ACM. https://doi.org/10.1145/1151954.1067473
[35] Georgantaki, S., & Retalis, S. (2007). Using educational tools for teaching object-oriented design and programming. *Journal of Information Technology Impact*, *7*(2), 111-130.
[36] Kelleher, C., & Pausch, R. (2005). Lowering the barriers to programming: A taxonomy of programming environments and languages for novice programmers. *ACM Computing Surveys* (*CSUR*), *37*(2), 83-137. https://doi.org/10.1145/1089733.1089734
[37] Chou, C. Y., Huang, B. H., & Lin, C. J. (2011). Complementary machine intelligence and human intelligence in virtual teaching assistant for tutoring program tracing. *Computers and Education*, *57*(4), 2303-2312. https://doi.org/10.1016/j.compedu.2011.06.005

9 Authors

Ming-Che Lee (leemc@mail.mcu.edu.tw) works at the Ming Chuan University, Department of Computer and Communication Engineering, Taiwan (R.O.C).

Ming Che Lee is an Associate Professor at Computer and Communication Engineering of Ming Chuan University (Taoyuan, Taiwan). His research interests are in the fields of Semantic Web, Deep Learning, Digital Content, and e-Learning Technologies.

Jia-Wei Chang (Corresponding Author jiaweichang.gary@gmail.com) works for The National Taichung University of Science and Technology, Department of Computer Science and Information Engineering, Taiwan (R.O.C)

Jia-Wei Chang is an Assistant Professor in Department of Computer Science and Information Engineering at National Taichung University of Science and Technology. Since January 2019, he is a Young Professionals Chair of the Institution of Engineering and Technology (IET) - Taipei Network.

Tzone I Wang (wti535@mail.ncku.edu.tw) is with The National Cheng Kung University, Department of Engineering Science, Taiwan (R.O.C)

Tzone I Wang is a Professor of Engineering Science at National Cheng Kung University, Taiwan.

His major research areas include Artificial Intelligence, Web and Network Services.

Zi Feng Huang (huang160812@gmail.com) is with National Cheng Kung University, Department of Engineering Science.

Zi Feng Huang is a master student in the Engineering Science at National Cheng Kung University, Taiwan.

Article submitted 2020-03-11. Resubmitted 2020-04-19. Final acceptance 2020-04-22. Final version published as submitted by the authors.