Article

Improving Undergraduate Novice Programmer Comprehension through Case-Based Teaching with Roles of Variables to Provide Scaffolding

Nianfeng Shi

School of Computer and Information Engineering, Luoyang Institute of Science and Technology, Luoyang 471000, China; shinf@lit.edu.cn

Abstract: A role-based teaching approach was proposed in order to decrease the cognitive load placed by the case-based teaching method in the undergraduate novice programmer comprehension. The results are evaluated by using the SOLO (Structure of Observed Learning Outcomes) taxonomy. Data analysis suggested novice programmers with role-based teaching tended to experience better performances, including the SOLO level of program comprehension, program debugging scores, program explaining scores, except for programming language knowledge scores, compared with the classical case-based teaching method. Considering the SOLO category of program comprehension and performances, evidence that the roles of variables can provide scaffolding to understand case programs through combining its program structure with its related problem domain is discussed, and the SOLO categories for relational reasoning are proposed. Meanwhile, the roles of variables can assist the novice in learning programming language knowledge. These results indicate that combing case-based teaching with the role of variables is an effective way to improve novice program comprehension.

Keywords: undergraduate novice programmer; program comprehension; case-based teaching; roles of variables; SOLO taxonomy

check for updates

Citation: Shi, N. Improving Undergraduate Novice Programmer Comprehension through Case-Based Teaching with Roles of Variables to Provide Scaffolding. *Information* **2021**, *12*, 424. https://doi.org/10.3390/info12100424

Academic Editor: Willy Susilo

Received: 4 September 2021
Accepted: 14 October 2021
Published: 16 October 2021

Publisher's Note: MDPI stays neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Copyright: © 2021 by the author. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (https://creativecommons.org/licenses/by/4.0/).

1. Introduction

As a novice, computer programming learning is a complex task [1,2]. Approximately 26.4% of computer science students from the Luoyang Institute of Science and Technology (LIT) did not pass the C language programming subject in the first semester from 2007 to 2014 [3]. Data from all over the world also indicate that more and more students do not want to pursue a major in computer programming in higher education [4]. One reason why students fail to learn programming might be that novice programmers struggle to perform relational reasoning [5] due to some misconceptions about variables [6]. As a result, they are usually either unable to accurately understand the fundamentals of programming [2] or combine the various statements and structures of the programming language into a valid program [7].

Programming is one cognitive task. Programmers require logical thinking together with technical skills. Over the past decades, many instructional methods and pedagogical techniques have been proposed to improve students' programming learning, such as creating games, live coding [8], case-based programming [9], and so on. Despite the differences among these teaching methods, most share the same basic point view: assisting the novice in program comprehension. The engagement with relational reasoning explicitly through program comprehension, e.g., reading, explaining, and debugging, could significantly improve novice program skills [10,11].

Concerning the most effective way to teach novices program comprehension, the case-based teaching method is usually used, which is a highly adaptable style of teaching that involves problem-based learning [12]. Previous studies have shown that case-based teaching can cultivate a higher level of critical thinking and foster creativity [4] and problem-solving skills [13]. These capabilities are necessary for novice programmers to comprehend programs. However, understanding the structure of the case programs, discussing and debating problem-solving strategies, and relating case problems to relevant theoretical knowledge all impose cognitive loads on novice programmers. Various pedagogical techniques, such as visualization, creative interaction, and animation, were utilized to assist novice programmers in literature program codes. However, the program comprehension is still challenging.

Prior research shows that the roles of variables enable novice programmers to engage in programming in a comprehensive and expert way [3,4,14]. Understanding programs is crucial to program comprehension. If we provide novice programmers with the roles of variables, they will understand the case program reasonably and improve their problem-solving performance, which in turn improves their programming learning outcomes.

We carried out one experiment on program comprehension in the C language programming course with the case-based teaching method to investigate our hypothesis. The SOLO (Structure of Observed Learning Outcomes) taxonomy was adopted to test whether the roles of variables can provide scaffolding to enhance the pedagogical effect.

2. Literature Review

2.1. Case-Based Teaching

Case-based teaching is a term of practical teaching approach, and it has been used successfully from as early as 1870 [15]. A case is a way to bridge the abstract nature of principles and teaching standards to classroom practice [12]. By using cases, the teacher can present students with examples or best practice models which can describe theoretical principles in practice. Previous studies have shown that case-based teaching can foster students' ability to analyze complex problems and to practice decision-making through group discussion. Meanwhile, case-based teaching can increase the student engagement in class activities, which promotes learning and increases achievements [16–18].

Case-based teaching has been successfully used in computer programming courses [19,20], and it is useful for deepening the understanding of the meaning of the concepts and to improve the design of the program [21]. Moreover, the effectiveness of case-based teaching in computer programming language is not related to entry qualifications [22].

However, learning about computer programming with the case-based teaching method is a big challenge to novice programmers. Novice programmers typically lack computer programming theoretical knowledge and programming experience. Case-based teaching, on the other hand, relies heavily on discussion and debate of the case issues and the array of potential solutions [23]. To discuss and debate on a case program, students need to understand the structure of the program and to relate it to relevant experiential and theoretical background. The teacher should carefully structure case-based instruction of novice students in order not to hinder acquisition of introductory domain knowledge [24]. Traditional case-based teaching with statics materials fails to teach program learning due to students' poor program understanding [25]. A dynamic learning environment, for example, education games [26], should be employed to develop students' mental representation of the problem [22,26,27]. Higher-level knowledge beyond the syntax and operation of computer programming is also needed to enable students to comprehend the case program at the early stage of case-based programming learning [28]. In brief, how to alleviate the cognitive loads placed by the case-based teaching method is still a hot topic to resolve in novice program learning.

2.2. Roles of Variables

Roles of variables are a conceptual framework that can be utilized to improve novice program learning. The concept of the roles of variables was introduced by [29] after he found: (a) variables used in programs have several standard use patterns and (b) 99% of variables in novice-level procedural programs could be covered by only ten roles. A role is characterized by the sequence of continuous values of a variable and its dependency on other variables, rather than the way that the variable is used [29]. There are eleven kinds of roles of variables in the novice C language programming, including *Fixed, Value, Temporary, One-way flag*, *Most-wanted holder*, *Most-recent holder*, *Gatherer*, *Organizer*, *Container*, *Follower*, *Stepper*, and *Walker* [14].

Roles of variables represent programming knowledge at a level higher than simple programming language knowledge [30]. Therefore they can be explicitly taught to students. Prior researches suggested that roles of variables could promote the programming learning of novice programmers and help students to comprehensive programming [31,32]. Students with the role-based teaching method tended to stress deep program structures as experienced experts while they had fewer problems with the adoption of variables in program learning [33].

Can roles of variables be employed by the program comprehension with the case-based teaching method in order to help novice programmers perform the discussion and debate of the case issues and the array of potential solutions? This is a fresh idea that can be adopted to alleviate the cognitive loads placed by the case-based program learning.

2.3. SOLO Taxonomy for Program Comprehension

SOLO (Structure of Observed Learning Outcomes) taxonomy presents a way to describe the increasing complexity of learners' activities. SOLO is one of the most commonly used taxonomies, which is based on the integration of details into a structural pattern (qualitative) and the amount of learners' knowledge (quantitative) [34]. Using this taxonomy, students' responses will be classified according to the level of integration rather than absolute correctness [35].

SOLO can be adopted to reliably encode student responses for program comprehension questions [34,36,37]. There are three reasons, including (a) SOLO enable a teacher examine how well a student can read several lines of code and integrate them into a valid program [37]; (b) SOLO is a useful organizing framework for comparing and evaluating work related to the assessment of novice programmer through reading problems [38]; and (c) the SOLO categories are consistent with the performance of students in program comprehension [36].

Program comprehension is a cognitive process. To understand how to better program, both the holistic point of view and the local perspective are required, which have students see both "the trees" and "the forest" [5]. The SOLO categories schema of program comprehension proposed in [35] (see Table 1) was used in this study, which has proven to be the most accurate and effective educational taxonomy for evaluating programming understanding [10,34,38,39].

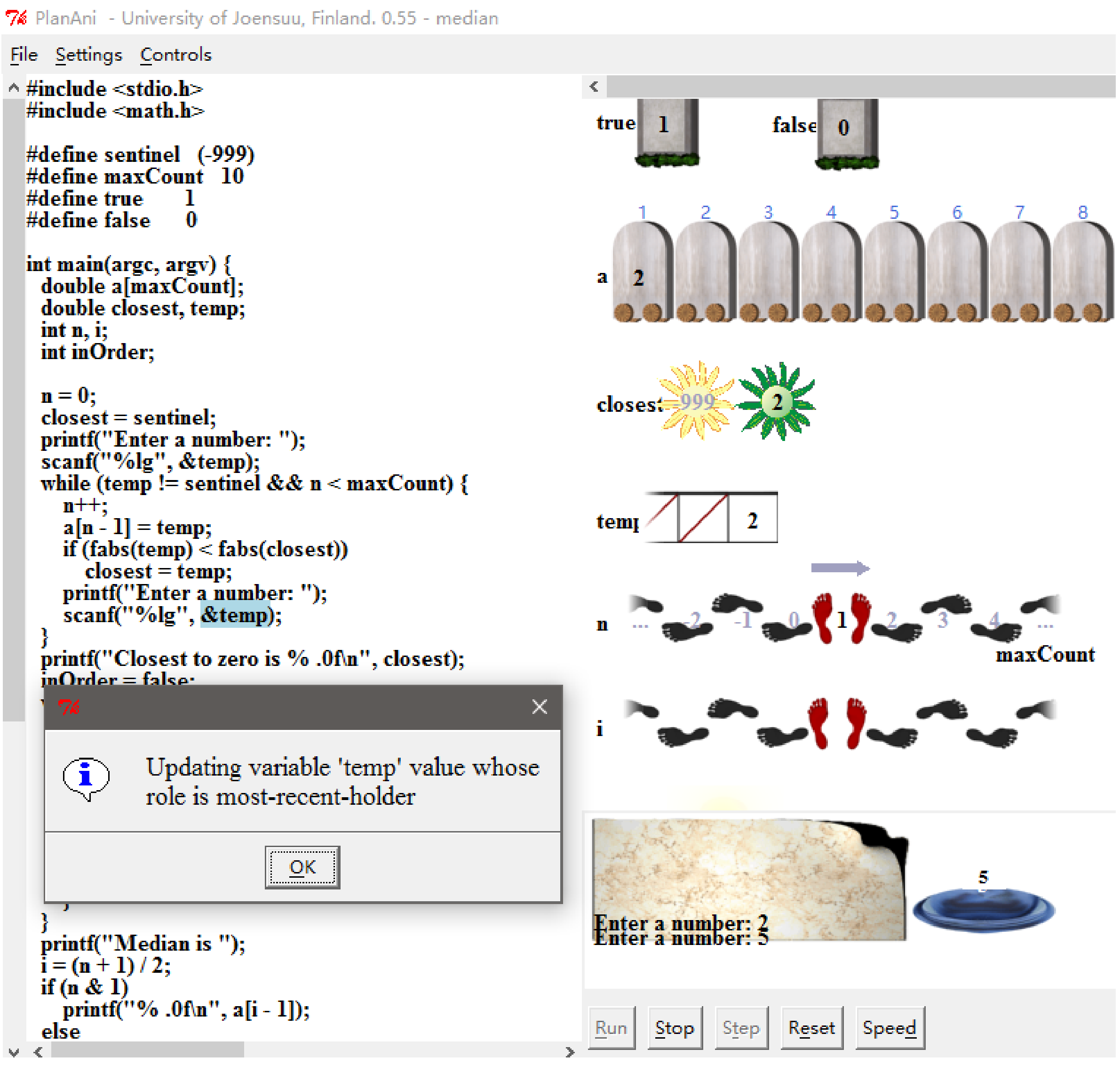
Table 1. SOLO categories for program comprehension [35].

SOLO Category	Description
Relational (R)	A summary of what the code does in terms of its purpose (the forest)
Relational Error (RE)	A summary of what the code does in terms of its purpose, but with some minor error
Multistructural (M)	A line-by-line description of all the code (the trees)
Other (O)	Any other description of part or all of the code, displaying no real evidence of understanding of the code as a whole

2.4. PlanAni System

It is a great challenge job for novice programmers to identify roles of variables and following program execution. Fortunately, the program visualization is a productivity tool that can increase roles of variables teaching effectiveness. In this investigation, a visualization tool named PlanAni system was employed.

PlanAni system is a productivity tool supporting the concept of the roles of the variable (see Figure 1). In PlanAni system, each variable in the computer program is visualized as a role image. For example, variable “*false*” is a role of *fixed value*, and is visualized as a stone in Figure 1, giving the impression that the value of a role of *fixed value* is initialized and cannot be changed. In addition to role images, role character animations are also employed to visually demonstrate the operations of variables by PlanAni system. For example, the animation of assignment to a *stepper* is that a footprint rolls smoothly along the footsteps, which implies the fact that the new values of steppers are known to be a succession of values from the beginning [30].



```
#include <stdio.h>
#include <math.h>

#define sentinel  (-999)
#define maxCount  10
#define true      1
#define false     0

int main(argc, argv) {
  double a[maxCount];
  double closest, temp;
  int n, i;
  int inOrder;

  n = 0;
  closest = sentinel;
  printf("Enter a number: ");
  scanf("%lg", &temp);
  while (temp != sentinel && n < maxCount) {
    n++;
    a[n - 1] = temp;
    if (fabs(temp) < fabs(closest))
      closest = temp;
    printf("Enter a number: ");
    scanf("%lg", &temp);
  }
  printf("Closest to zero is % .0f\n", closest);
  inOrder = false;
```

Updating variable 'temp' value whose role is most-recent-holder

OK

```
    }
  }
  printf("Median is ");
  i = (n + 1) / 2;
  if (n & 1)
    printf("% .0f\n", a[i - 1]);
  else
```

```
Enter a number: 2
Enter a number: 5
```

Figure 1. The interface of the PlanAni system.

3. Research Method

3.1. Participants

A total of 57 first-grade undergraduate students were enrolled in this research, all of whom were majoring in computer science and technology from the Luoyang Institute of Science and Technology (LIT). They all attended the C Language Programming Course in the second semester of the 2018/2019 school year, which is a mandatory subject with a total of 36 class hours, including 8 laboratory exercises and 28 theoretical lectures.

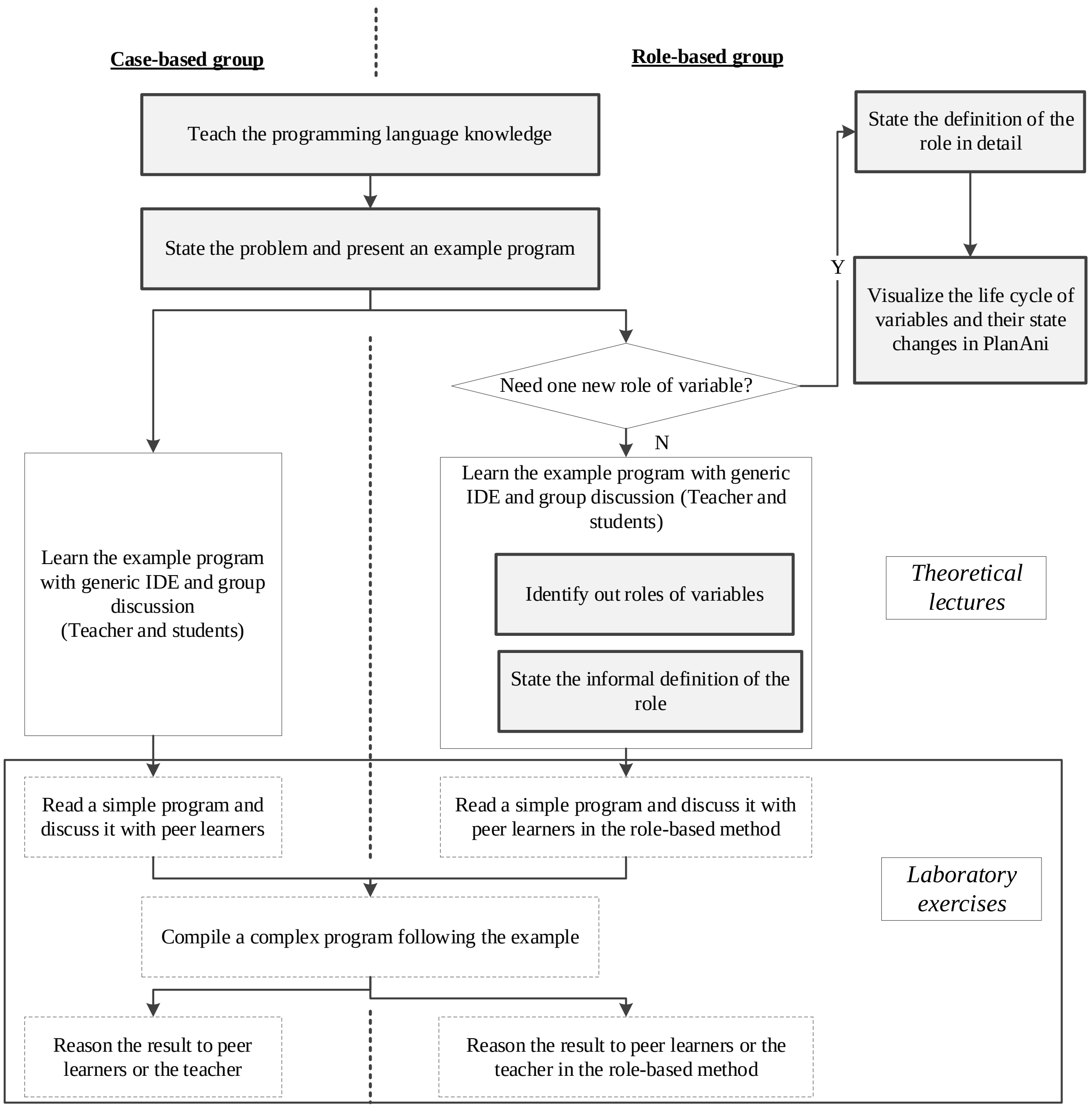
One control (N = 27) and one experimental (N = 30) group were used in this study, which were randomly assigned based on the two intact classes. Therefore, nonequivalent pre-test post-test control group design was used.

3.2. Instruments

Three types of instruments, SOLO levels of the program comprehension, case-based teaching method, and PlanAni system were used in the study.

3.3. Procedure

The experiment started from the first week of the second semester of the 2018/2019 school year, lasting over eighteen weeks with two class hours per week. Experimental group students were instructed with a revised case-based teaching approach supported by roles of variables (named role-based teaching method) in learning the C Language Programming Course. In contrast, control group students used the classical case-based teaching method. Figure 2 is the diagram of the educational activities in the traditional group (left side) and the role-based group (right side). Activities of the theoretical lectures are depicted in the upper part of Figure 2, and laboratory exercises are in the lower one. Activities with dotted borders are conducted by the teacher, activities with dashed borders are carried out by students, and the others are completed by the teacher and students together.



Case-based group
Role-based group
Teach the programming language knowledge
State the problem and present an example program
Need one new role of variable?
Y
N
State the definition of the role in detail
Visualize the life cycle of variables and their state changes in PlanAni
Learn the example program with generic IDE and group discussion (Teacher and students)
Learn the example program with generic IDE and group discussion (Teacher and students)
Identify out roles of variables
State the informal definition of the role
Theoretical lectures
Read a simple program and discuss it with peer learners
Read a simple program and discuss it with peer learners in the role-based method
Laboratory exercises
Compile a complex program following the example
Reason the result to peer learners or the teacher
Reason the result to peer learners or the teacher in the role-based method

Figure 2. Pedagogical activities in the control and experimental groups.

Before the semester, the researchers collected some case programs and made them into handouts. In addition to the handouts, the definitions of the eleven roles with short program examples in C language described in [40] were printed as booklets to conduct the role-based teaching in Chinese. At the beginning of the course, all students were presented with the printed handouts. However, the booklets on the roles of variables were only handed out to students from the role-based group.

The theoretical lectures and laboratory exercises of the course were methodically conducted. The pedagogical resources, such as case programs, laboratory tasks, homework assignments, and the sum of teaching time, were the same in the experimental and control groups, but roles of variables were only carried out in the experimental group (see Figure 2).

The C Language Programming Course in the experimental group was conducted as follows. Firstly, the role of the variable was gradually presented in the theoretical lectures (see Figure 2). The teacher provided some basic concepts of C programming language covered by case programs. As a new role appearing in the case program, the teacher instructed its definition in detail with the role-based animation and provided students with a case program in the PlanAni system (the more information about PlanAni can be found in the coming related contents). Thereafter, the teacher will repeat the informal definition and characteristics of a role whenever it reappeared in the forthcoming case programs. In practice, the code tracing and debugging technologies, supported by the generic visual IDE (Integrated Development Environment), were employed to visually show the life cycle of the variable and intuitively elaborate the function of a role. The teacher set breakpoints in the program codes and systematically explained the sequence of states of each variable role and its dependency on other variables with the visual debugging tools of IDE, such as *QuickWatch, Step over, Set Next Statement*, and so on. During the process, the teacher and students predicted the following states of a variable and the other variables it depended on, viewed its execution process, and checked the final execution result. To deepen and consolidate students' understanding of the roles of a variable, in the next two homework assignments, participants must outline the role and reason for its adoption.

Secondly, the role of the variable was systematically practiced in the laboratory exercises (see Figure 2). Three educational activities were required to make students exactly understand the roles of variables and appropriately apply them to the program comprehension: (a) students were asked to read a straightforward case program carefully selected by the teacher for 10 to 15 min; (b) students compiled a complex computer program by revising the example program; and (c) students were required to identify the roles in their program and then explain the life cycle of the variable to the teacher or their peer students with the visual IDE as demonstrating the results.

At the last stage of the course, all participants were asked to complete a final examination. The final exam was a two-hour paper-and-pencil test including four types of items: basic knowledge questions (including the true or false questions (TFQ) and multiple-choice questions (MCQ)), program explaining (EXPL), program debugging (DBG), and program construction (CONS).

3.4. Data Collection

Two variable data sets were collected: the final paper-and-pencil test results and SOLO levels of the program comprehension.

Previous studies suggested that the SOLO level of the responses to program explanation was consistent with program comprehension. There were two steps to obtain the SOLO level of program comprehension. First, a participant's answers to EXPL were assigned SOLO levels according to Table 1. Then, the SOLO level received was converted into corresponding scores. Table 2 is the numeric SOLO categories of program comprehension adopted in the present study.

Table 2. Numeric SOLO categories of program comprehension.

SOLO Category	Converted Score	SOLO Category	Converted Score
R	4	M	2
RE	3	O	1

4. Results and Findings

An independent *t*-test of the final paper-and-pencil test scores of the two groups was conducted to evaluate the overall effectiveness of the role-based teaching method. As shown in Table 3, there was a significant effect for the teaching method, $t(2.61) = 55.0$, $p = 0.012$, with students from the experimental group receiving higher scores for the final paper-and-pencil test than from the control group. The mean of final paper-and-pencil test scores in the experimental group was 10% more than the control group, suggesting the role-based teaching method was more effective than the classical case-based teaching approach.

Table 3. Results of the *t*-test of the final paper-and-pencil test scores.

Group	N	M	SD	*t*	df	*p*
Experimental	27	79.7	10.01	2.61	55	0.012
Control	30	71.8	12.65			

Some data analyses were carried out to understand much more about the difference, especially the effects of role-based teaching on novice program comprehension. Firstly, a Mann-Whitney U test of the scores for TFQs and MCQs was conducted because these scores were not normally distributed. Table 4 showed that the score of TFQs and MCQs in the experimental (Mdn = 32) group was the same in the control group (Mdn = 33.5), U = 509.0, $p = 0.095$. TFQs and MCQs are related to the programming language knowledge of C language, which is the basis for program comprehension. Given the exact distribution of the scores of TFQs and MCQs achieved, it will imply that the learning of the programming language knowledge about C language was sufficiently carried out in the experimental and control groups.

Table 4. Results of the Mann-Whitney U test of the scores for TFQs and MCQs.

Group	N	MR	U	z	*p*
Experimental	27	25.15	509.0	1.67	0.095
Control	30	32.47			

Secondly, to evaluate the effect of the role-based teaching method on the ability to comprehend the program, a Mann-Whitney U test of the SOLO scores of responses for EXPL, between the experimental and control groups was used due to the non-normal distribution of the numeric SOLO scores. Table 5 showed that the SOLO score of response for EXPL was higher in the experimental (Mdn = 3.5) than in the control group (Mdn = 2.5), U = 263.5, $p = 0.019$. In Figure 3, 37% of the students from the experimental group achieved a whole SOLO level of program comprehension compared to 17% of the control group. These results indicated that students with role-based teaching tended to understand a program from a relational view corresponding to the control group.

Table 5. Result of the Mann-Whitney U test of the SOLO scores of responses for EXPL.

Group	N	MR	U	z	*p*
Experimental	27	32.24	263.5	−2.36	0.019
Control	30	24.28			

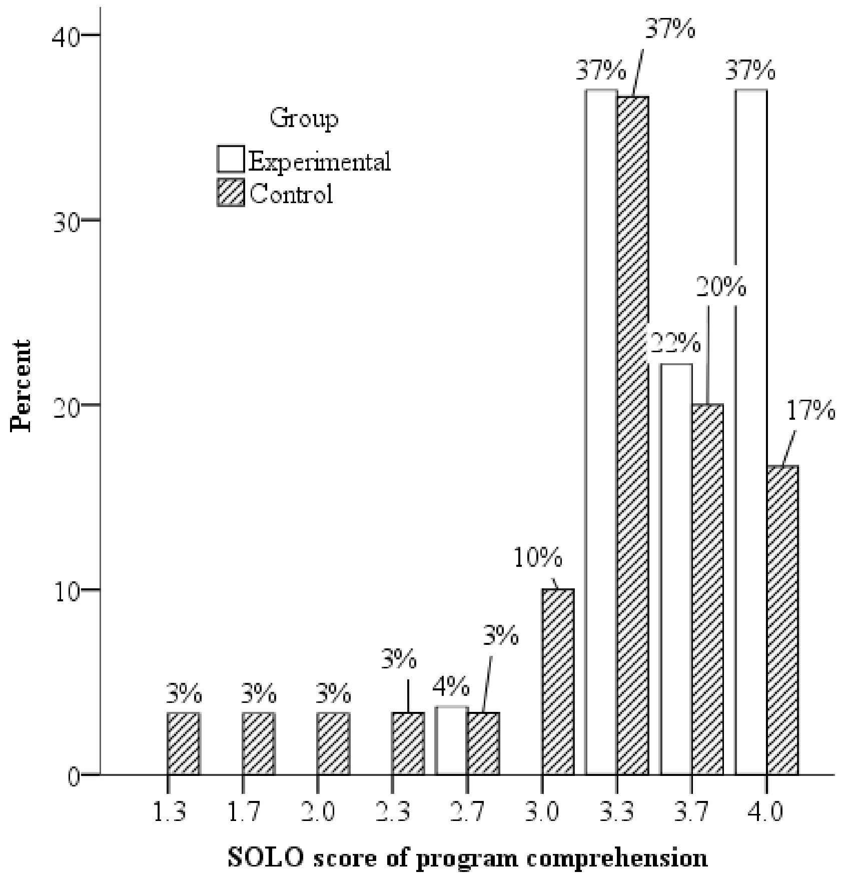


Figure 3. Distribution of the SOLO scores of program comprehension.

Lastly, an independent *t*-test of the EXPL scores between the experimental and control groups was used to test whether the students from the role-based group performed better in EXPL than the classical case-based group. Table 6 showed that the EXPL scores in the experimental group were significantly higher, $t(2.72) = 55$, $p = 0.009$, than in the control group. The mean EXPL score in the experimental group was 9% higher than the control group. The results indicate that the role-based group outperformed the traditional group in EXPL.

Table 6. Results of the *t*-test of the EXPL scores.

Group	N	M	SD	*t*	df	*p*
Experimental	27	24.0	3.45	2.72	55	0.009
Control	30	21.3	3.98			

Furthermore, to test the effectiveness of the role-based teaching method in program debugging, a Mann-Whitney U test of the DBG scores was also conducted between the experimental and control groups. Table 7 showed the DBG score was higher in the experimental group (Mdn = 8) than in the control group (Mdn = 5.5), U = 164.5, $p < 0.001$. Meanwhile, the scores of DBG of the experimental group were between 8 and 10 (about 67%), while between 4 and 6 (about 67%) in the control group (see Figure 4). These results suggested that the students in the experimental group received better scores of DBG than the control group.

Table 7. Result of the Mann-Whitney U test of the DBG scores.

Group	N	MR	U	z	*p*
Experimental	27	37.91	164.50	−3.89	<0.001
Control	30	21.00			

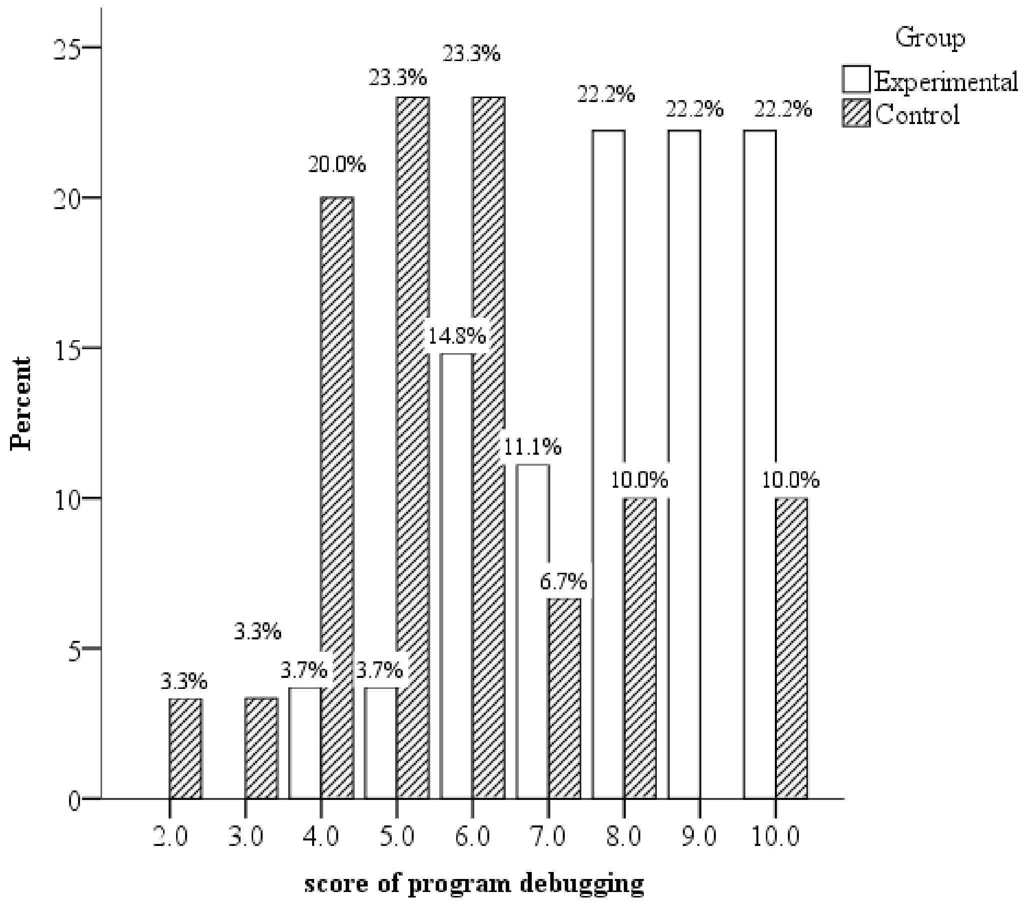


Figure 4. Distribution of program debugging scores.

5. Discussion

This research reveals that the role-based teaching method can yield better achievements than the classical case-based teaching approach. In the final exam, 11% of students in the role-based group achieved the same or higher than 90% accuracy, compared to 5% in the classical case-based group. In the program explaining, the mean score was increased by 9% by using the role-based teaching method. Meanwhile, in program debugging, 22% of students in the role-based group received full marks, compared to 10% of the classical case-based group, as shown in Figure 4. Apart from programming knowledge, practical skills are essential to novice programmers. Case-based teaching can help programmers go beyond program comprehension to integrate programs with programming language syntax and semantics [21]. Roles of variables can teach novice programmers to understand computer programming [14]. Studies suggested that ‘understanding’ and ‘application’ can improve students’ intentions to learn programming, and motivation can significantly enhance students’ performance [2,7,21,22].

This research also suggests that the students with role-based teaching tend to present relational answers to the program explanations, which helps them to become experts. In Figure 3, students with SOLO scores of program comprehension of 3.3, 3.7, and 4.0 accounted for 37%, 22%, and 37% of the total number of students in the role-based group, compared to 37%, 20%, and 17% of the classical case-based group. The SOLO score of program comprehension of 3.3, 3.7, and 4.0 mean that students scored out of one, two, or three full SOLO scores of responses for EXPL. According to Tables 1 and 2, a full SOLO score of program comprehension indicates that a novice programmer can read the code reasonably and systematically and extract their objectives. Expert programmers were good at presenting their program summaries by concentrating on both language-level concepts and problem domain-level concepts [41]. Thus, these results imply that the roles of variables can help the novice read several lines codes and integrate them into a coherent structure [33] like an expert.

This research show that instead of using the classic case-based teaching method focusing on programming language syntax and semantics or example programs, the teacher might utilize the roles visualization and program explaining to teach the student programming language knowledge about C language. There was no significant difference in their scores of TFQ and MCQ between the experimental and control groups. We did not expect this result. In addition to the C language programming, in the case of the same length of time, the students from the experimental group also needed to learn the concept of the roles of variables. Identifying the roles of variables and following program execution are not easy tasks for novice programmers, so students need to devote more time to learn about the roles of variables. The following reasons might be involved to explain this result. Firstly, visualizing roles information improved the learning of the programming language knowledge about C language. In this research, the role-based animations by the PlanAni system were adopted to help the novice learn the concept of the roles of variables. Roles of variables, on the other hand, are related to programming language knowledge, and they always be instructed with example programs. By understanding the operation and adoption of the roles of variables, the novice can know what a variable is, and how it is used in a program. For example, by watching the animation of *stepper* and learning the concept of *stepper*, the novice can know what the loop structure is and how it works in a program. Comparing to the learning with the text only, visualizing the loop structure by the role-based animation is more effective. This opinion supports that visualization techniques can motivate students to engage in programming and help them learn significantly more programming concepts [2,42].

Secondly, program explaining assisted the novice in learning the programming language knowledge about C language. Programming learning is highly complex, including acquiring programming knowledge and developing programming skills [43]. Even at the level of computer literacy, it requires the construction of conceptual knowledge and the structuring of basic operations (such as loops, conditional statements, etc.) into schema and plans. In this research, experimental group students were required to list all roles in their homework. In the laboratory exercise, participants were asked to learn conceptual program knowledge by reasoning and to explain the roles of variables. Some studies reported that program explaining and reasoning could significantly improve novice programmers' understanding of abstract program language concepts [44,45].

Furthermore, this article implies that the grade and SOLO taxonomy can effectively evaluate the educational achievement as long as the assessment model is appropriate. Programming is a complex cognitive process. A teacher must select the appropriate educational assessment tools to accurately evaluate students' learning achievements and conduct the programming teaching work efficiently during the course. In this research, we noticed that the effects of the role-based teaching method can be reflected not only by either the SOLO level of program comprehension but also by the accuracy of program debugging and program explaining. This finding suggests that, during the learning proceeding, the teacher can use the SOLO category to analyze students' level of program comprehension, helping them design a personalized learning path.

Meanwhile, the teacher can also employ the final scores for program comprehension to test the ability of the novice to understand programs. This finding stands on the opposite side of [33], which reported that students' grades could not improve program comprehension by the role-based teaching. The following reason can be used to make our results sense. Grading was found to reflect teachers' expectations of "quality" answers [33].

Based on these results, we can conclude that the roles of variables provide scaffolding to assist novice undergraduate programmer comprehension with case-based teaching effectively. The SOLO taxonomy can be used to assess the ability of the novice to understand a program.

6. Conclusions and Recommendations

6.1. Conclusions

Learning program comprehension is a big challenge to novice programmers. Our findings demonstrate that the roles of variables can significantly improve the novice programmer's comprehension of C language with the case-based teaching method. Data analysis suggests that the improvement contributes to the SOLO level of program comprehension, debugging scores, and explaining scores, but no programming language knowledge scores.

Meanwhile, this study shows:

(1) The roles visualization and program explaining can play in teaching the student programming language knowledge instead of focusing on programming language syntax and semantics.

(2) Students with role-based teaching tend to present relational responses to program comprehension, making them compile complex programs like experienced programmers.

6.2. Recommendations

In this section, we present some practical suggestions on roles of variables.

A teacher can directly employ the SOLO categories and PlanAni system adopted in this study to teach novice programmers C language learning. However, the roles of variables in other programming should be reorganized and designed because the programming knowledge, especially in the operation of variables, is very different [4]. Case programs should be carefully selected to cover all roles of variables.

The latest study shows that the home language has an effect on enhancing outcomes of programming learning. We suggest that a teacher can translate the definition of roles and variables into the home language [2].

In this study, all teachers agree that the more general the answer to program explaining is, the higher the ability. Thus, teachers in our study gave better grades for superior program comprehension skills. We think that the SOLO taxonomy can be utilized to evaluate students' ability to deploy relational reason in the other subjects. Therefore, we extended the SOLO categories for relational reasoning as shown in Table 8.

Table 8. SOLO categories for relational reasoning tasks.

SOLO Category	Description
Relational (R)	Students associate the concept/process and combine them into relevant conclusions (the forest)
Relational Error (RE)	Students associate the concept/process and combine them into relevant conclusions, but with some minor errors
Multistructural (M)	Students use the concept/process, but find no relationship between the information, so the conclusions are irrelevant (the trees)
Other (O)	Any other description of part or all of the concept/process, displaying no real evidence of understanding of the piece of information

7. Limitations and Future Work

One of the major limitations in the present study is that we did not measure the program comprehension using web-based learning. Web-based programming learning will pressure students and discourage novice programmers due to its poor interactivity [2]. The cognitive loads of programming learning placed by web-based learning are higher than by face-2-face teaching. Future work could conduct such experiments to examine the effectiveness of the roles of variables.

Another limitation of our research is the number of paper-and-pencil tests. We only carried out one final paper-and-pencil test. Therefore, our findings cannot measure the dynamic effectiveness of improving novice programmers' program comprehension. Future work could increase the times of trials to test the effectiveness of the roles of variables in the different stages of programming learning.

Funding: This research was funded by RESEARCH AND PRACTICE PROJECT OF HIGHER EDUCATION TEACHING REFORM OF HENAN PROVINCE, CHINA, grant number 2019SJGLX457.

Acknowledgments: The author would like to thank the anonymous reviewers for their valuable feedback. They also thank Yajun Pang for her English proofreading and assistance with data collection.

Conflicts of Interest: The author declares no conflict of interest.

References

1. Kuittinen, M.; Sajaniemi, J. Teaching roles of variables in elementary programming courses. *SIGCSE Bull.* **2004**, *36*, 57–61. [CrossRef]
2. Prasad, A.; Chaudhary, K.; Sharma, B. Programming skills: Visualization, interaction, home language and problem solving. *Educ. Inf. Technol.* **2021**. Available online: https://link.springer.com/article/10.1007/s10639-021-10692-z?utm_source=xmol&utm_medium=affiliate&utm_content=meta&utm_campaign=DDCN_1_GL01_metadata (accessed on 13 October 2021). [CrossRef]
3. Shi, N.; Min, Z.; Zhang, P. Effects of visualizing roles of variables with animation and IDE in novice program construction. *Telemat. Inform.* **2017**, *34*, 743–754. [CrossRef]
4. Papadakis, S. Evaluating a Teaching Intervention for Teaching STEM and Programming Concepts Through the Creation of a Weather-Forecast App for Smart Mobile Devices. In *Handbook of Research on Tools for Teaching Computational Thinking in P-12 Education*; IGI Global: Hershey, PA, USA, 2020; pp. 31–53.
5. Corney, M.; Lister, R.; Teague, D. Early relational reasoning and the novice programmer: Swapping as the "hello world" of relational reasoning. In Proceedings of the Thirteenth Australasian Computing Education Conference, Perth, Australia, 17–20 January 2011; pp. 95–104.
6. Kohn, T. Variable Evaluation: An Exploration of Novice Programmers' Understanding and Common Misconceptions. In Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education, Seattle, WA, USA, 8–11 March 2017; pp. 345–350.
7. Algaraibeh, S.M.; Dousay, T.A.; Jeffery, C.L. Integrated Learning Development Environment for Learning and Teaching C/C++ Language to Novice Programmers. In Proceedings of the 2020 IEEE Frontiers in Education Conference (FIE), Uppsala, Sweden, 21–24 October 2020; pp. 1–5.
8. Raj, A.G.S.; Gu, P.; Zhang, E.; Williams, J.; Halverson, R.; Patel, J.M. Live-coding vs Static Code Examples: Which is better with respect to Student Learning and Cognitive Load? In Proceedings of the Twenty-Second Australasian Computing Education Conference, Melbourne, Australia, 4–6 February 2020; pp. 152–159.
9. Shrestha, N.; Botta, C.; Barik, T.; Parnin, C. Here we go again: Why is it difficult for developers to learn another programming language? In Proceedings of the IEEE/ACM 42nd International Conference on Software Engineering (ICSE), Seoul, Korea, 5–11 October 2020; pp. 691–701.
10. Whalley, J.L.; Lister, R.; Thompson, E.; Clear, T.; Robbins, P.; Kumar, P.K.A.; Prasad, C. An australasian study of reading and comprehension skills in novice programmers, using the bloom and SOLO taxonomies. In Proceedings of the 8th Australasian Conference on Computing Education, Hobart, Australia, 16–19 January 2006; pp. 243–252.
11. Shargabi, A.A.; Aljunid, S.A.; Annamalai, M.; Zin, A.M. Performing tasks can improve program comprehension mental model of novice developers: An empirical approach. In Proceedings of the 28th International Conference on Program Comprehension, Seoul, Korea, 13–15 June 2020; pp. 263–273.
12. Bonney, K.M. Case study teaching method improves student performance and perceptions of learning gains. *J. Microbiol. Biol. Educ.* **2015**, *16*, 21–28. [CrossRef] [PubMed]
13. Gravett, S.; de Beer, J.; Odendaal-Kroon, R.; Merseth, K.K. The affordances of case-based teaching for the professional learning of student-teachers. *J. Curric. Stud.* **2016**, *49*, 1–22. [CrossRef]
14. Shi, N.; Cui, W.; Zhang, P.; Sun, X. Evaluating the effectiveness roles of variables in the novice programmers learning. *J. Educ. Comput. Res.* **2018**, *56*, 181–201. [CrossRef]
15. Merseth, K.K. The early history of case-based instruction: Insights for teacher education today. *J. Teach. Educ.* **1991**, *42*, 243–249. [CrossRef]
16. Flynn, A.E.; Klein, J.D. The influence of discussion groups in a case-based learning environment. *Educ. Technol. Res. Dev.* **2001**, *49*, 71–86. [CrossRef]
17. Baker, E.B.A. Multimedia case-based instruction in literacy: Pedagogy, effectiveness, and perceptions. *J. Educ. Multimed. Hypermedia* **2009**, *18*, 249–266.
18. Luo, H. Applying the case-based method in designing self-directed online instruction. *Diss. ALL* **2015**, *254*. Available online: https://surface.syr.edu/cgi/viewcontent.cgi?article=1254&context=etd (accessed on 13 October 2021).

19. Marks, J.; Freeman, W.; Leitner, H. Teaching applied computing without programming: A case-based introductory course for general education. *SIGCSE Bull.* **2001**, *33*, 80–84. [CrossRef]
20. Liu, G.; Yang, Q.; Fan, R. Application of case-based teaching in higher vocational computer courses—A case study of delphi programming. In Proceedings of the 2nd International Conference on Soft Computing in Information Communication Technology, Taipei, China, 31 May–1 June 2014; Atlantis Press: Taipei, China, 2014.
21. Chang, C.-S.; Chung, C.-H.; Chang, J.A. Influence of problem-based learning games on effective computer programming learning in higher education. *Educ. Technol. Res. Dev.* **2020**, *68*, 2615–2634. [CrossRef]
22. Veerasamy, A.K.; D'Souza, D.; Lindén, R.; Laakso, M.J. Relationship between perceived problem-solving skills and academic performance of novice learners in introductory programming courses. *J. Comput. Assist. Learn.* **2019**, *35*, 246–255. [CrossRef]
23. Apeanti, W.O.; Essel, D.D. Learning Computer Programming Using Project-Based Collaborative Learning: Students' Experiences, Challenges and Outcomes. *Int. J. Innov. Educ. Res.* **2021**, *9*. [CrossRef]
24. Demetriadis, S.; Pombortsis, A. Novice student learning in case based hypermedia environment: A quantitative study. *J. Educ. Multimed. Hypermedia* **1999**, *8*, 241–269.
25. Cheah, C.S. Factors contributing to the difficulties in teaching and learning of computer programming: A literature review. *Contemp. Educ. Technol.* **2020**, *12*, ep272. [CrossRef]
26. Mathew, R.; Malik, S.I.; Tawafak, R.M. Teaching Problem Solving Skills using an Educational Game in a Computer Programming Course. *Inform. Educ.* **2019**, *18*, 359–373. [CrossRef]
27. Robins, A.V. 12 Novice Programmers and Introductory Programming. In the Cambridge Handbook of Computing Education Research. 2019, p. 327. Available online: https://www.cambridge.org/core/books/abs/cambridge-handbook-of-computing-education-research/novice-programmers-and-introductory-programming/0CEDFE1B121198D3FB5F1541EBE3DCAD (accessed on 13 October 2021).
28. Bosse, Y.; Gerosa, M.A. Why is programming so difficult to learn? Patterns of Difficulties Related to Programming Learning Mid-Stage. *ACM SIGSOFT Softw. Eng. Notes* **2017**, *41*, 1–6. [CrossRef]
29. Sajaniemi, J. An Empirical Analysis of Roles of Variables in Novice-Level Procedural Programs. In Proceedings of the IEEE 2002 Symposia on Human Centric Computing Languages and Environments, Washington, DC, USA, 3–6 September 2002; pp. 37–39.
30. Sajaniemi, J.; Kuittinen, M. Visualizing roles of variables in program animation. *Inf. Vis.* **2004**, *3*, 137–153. [CrossRef]
31. Byckling, P.; Sajaniemi, J. Roles of variables and programming skills improvement. *SIGCSE Bull.* **2006**, *38*, 413–417. [CrossRef]
32. Al-Barakati, N.M.; Al-Aama, A.Y. The effect of visualizing roles of variables on student performance in an introductory programming course. *SIGCSE Bull.* **2009**, *41*, 228–232. [CrossRef]
33. Sajaniemi, J.; Kuittinen, M. An experiment on using roles of variables in teaching introductory programming. *Comput. Sci. Educ.* **2005**, *15*, 59–82. [CrossRef]
34. Clear, T.; Whalley, J.; Lister, R.; Carbone, A.; Hu, M.; Sheard, J.; Simon, B.; Thompson, E. Reliably classifying novice programmer exam responses using the SOLO taxonomy. In Proceedings of the 21st Annual Conference of the National Advisory Committee on Computing Qualifications, Auckland, New Zealand, 4–7 July 2008; pp. 23–30.
35. Lister, R.; Clear, T.; Simon; Bouvier, D.J.; Carter, P.; Eckerdal, A.; Jacková, J.; Lopez, M.; McCartney, R.; Robbins, P.; et al. Naturally occurring data as research instrument: Analyzing examination responses to study the novice programmer. *SIGCSE Bull.* **2010**, *41*, 156–173. [CrossRef]
36. Sheard, J.; Carbone, A.; Lister, R.; Simon, B.; Thompson, E.; Whalley, J.L. Going SOLO to assess novice programmers. In Proceedings of the 13th Annual Conference on Innovation and Technology in Computer Science Education, Madrid, Spain, 30 June–2 July 2008; pp. 209–213.
37. Lister, R.; Simon, B.; Thompson, E.; Whalley, J.L.; Prasad, C. Not seeing the forest for the trees: Novice programmers and the SOLO taxonomy. *SIGCSE Bull.* **2006**, *38*, 118–122. [CrossRef]
38. Seiter, L. Using SOLO to Classify the Programming Responses of Primary Grade Students. In Proceedings of the 46th ACM Technical Symposium on Computer Science Education, Kansas, MO, USA, 4–7 March 2015; pp. 540–545.
39. Izu, C.; Schulte, C.; Aggarwal, A.; Cutts, Q.; Duran, R.; Gutica, M.; Heinemann, B.; Kraemer, E.; Lonati, V.; Mirolo, C. Fostering program comprehension in novice programmers-learning activities and learning trajectories. In Proceedings of the Working Group Reports on Innovation and Technology in Computer Science Education, Aberdeen, UK, 15–17 July 2019; pp. 27–52. Available online: https://research.aalto.fi/en/publications/fostering-program-comprehension-in-novice-programmers-learning-ac (accessed on 13 October 2021).
40. Sajaniemi, J. Role List for Students (v2,C). Available online: http://www.cs.uef.fi/papges/saja/var_roles/stud_vers/stud_C_eng.html (accessed on 13 October 2021).
41. Pennington, N. Comprehension strategies in programming. In *Empirical Studies of Programmers: Second Workshop*; Olson, G.M., Sheppard, S., Soloway, E., Eds.; Ablex Publishing Corporation: Norwood, NJ, USA, 1987; pp. 100–113.
42. Kumar, A.N. The Effectiveness of Visualization for Learning Expression Evaluation: A Reproducibility Study. In Proceedings of the 2016 ACM Conference on Innovation and Technology in Computer Science Education, Arequipa, Peru, 11–13 July 2016; pp. 192–197.
43. Rogalski, J.; Samurçay, R. Acquisition of Programming Knowledge and Skills. In *Psychology of Programming*; Hoc, J.M., Green, T.R.G., Samurçay, R., Gillmore, D.J., Eds.; Academic Press: London, UK, 1990; pp. 157–174.

44. Kumar, A.N. A Study of the Influence of Code-Tracing Problems on Code-Writing Skills. In Proceedings of the 18th ACM Conference on Innovation and Technology in Computer Science Education, Canterbury, UK, 1–3 July 2013; pp. 183–188.
45. Busjahn, T.; Schulte, C. The use of Code Reading in Teaching Programming. In Proceedings of the 13th Koli Calling International Conference on Computing Education Research, Koli, Finland, 14–17 November 2013; pp. 3–11.