The Effect of Visualizing Roles of Variables on Student Performance in an Introductory Programming Course

Nouf M. Al-Barakati  
College of Computing and Information Technology  
King Abdul-Aziz University  
Jeddah, Saudi Arabia  
nalbarakati@kau.edu.sa

Arwa Y. Al-Aama  
College of Computing and Information Technology  
King Abdul-Aziz University  
Jeddah, Saudi Arabia  
aalaama@kau.edu.sa

ABSTRACT  
The use of variables in computer programming is one of the difficulties faced by students enrolled in introductory level programming classes. The Roles of Variables (ROV) concept associates small comprehensible roles to variables to help novice programmers comprehend how variables should be used. This paper describes an experiment that was conducted to test the effect of different engagement levels with ROV visualization on student programming skills. 91 Female students at King Abdul Aziz University (KAU), Jeddah, Saudi Arabia, participated in the experiment. The students were divided into three groups where they all had the ROV concept explained to them in a traditional classroom setting, but were given different visualization treatments during lab sessions. Results showed that while viewing the ROV visualization significantly improved student debugging skills, no other significant effects on student programming skills were reported.

Categories and Subject Descriptors  
D.3.3 [Programming Languages]: Language Constructs and Features, variables and data types  
K.3.2 [Computer and Information Science Education]: Computer science education, teaching/learning strategies

General Terms  
Measurement, Performance, Experimentation, Human Factors, Standardization, Languages

Keywords  
Roles of Variables, Visualization, Introductory Programming Course

1. INTRODUCTION  
Programming skills are essential to all Computer Science students. Students start constructing these skills in introductory programming courses. At King Abdul Aziz University (KAU), Computer Science students enroll in two introductory courses during their first year as Computer Science majors. In the first course, Computer Science 101 (CS101), students study "Fundamentals of Computer Science". In addition, they receive some introductory practical programming instruction using the C++ programming language. In the second course: "An Introduction to Computer Programming", or Computer Science 102 (CS102), students learn principles of programming and problem solving. They also learn syntax and semantics of the C++ programming language.

Research shows that novice programmers in C++ introductory programming courses confront comprehension difficulties as a result of using abstract concepts [14]. One of these abstract concepts is variables usage which is a tacit implicit knowledge that can not be presented explicitly to students [11]. Therefore, use of variables in programs is one of the weaknesses of novice programmers. However, it is essential for computer programming students to master this skill. Since variable usage can not be presented explicitly to students, students then need to mentally construct this programming knowledge from other types of knowledge by themselves; and sometimes may not construct such knowledge correctly. As a result, syntax and semantic errors are not identified, nor corrected quickly.

In 2002, Sajaniemi introduced the concept of the Roles of Variables (ROV) in which programming knowledge can be explicitly taught to students [10, 11]. Sajaniemi used ROV to represent schematic use of variables in programs by illustrating the dynamic nature of the consecutive values acquired by a variable, while the program is running. A role is not an exclusive task that is specific to certain programs but is a general concept that appears in programs continuously. Roles of Variables and their visualization are used in classrooms to enhance student learning [9, 11, 12, 13].

Computer Aided Instruction (CAI) and Computer Aided Learning (CAL) tools are being increasingly used in Computer Science Education [7]. Among these tools are visualization tools that are used for viewing graphical progression of programs or algorithms. They are powerful support tools for helping novices realize the behavior of computer programs [6]. The Engagement Taxonomy introduced by Naps et. al. [5] identifies six forms of learner engagement with visualization tools. The Taxonomy provides a framework to lead empirical experiments that evaluate instructional effectiveness of visualization.

In this research, it was anticipated to test whether integrating visualization tools with the use of Roles of Variables techniques may foster learning and increase novice performance and understanding. The research aimed to evaluate the effects of different engagement levels with ROV visualization on student learning in an introductory C++ programming course. The used visualization tool named the PlanAni. Students enrolled in the
This paper first provides an overview of the concept of Roles of Variables, a description of visualization and its engagement levels, and the PlanAni tool. Next, it describes the research methodology and presents some of the collected data. Finally, the paper presents key results, findings and conclusions.

2. ROLES OF VARIABLES

Variable usage is one type of programming knowledge that hinders student learning in introductory programming courses. Misuse of variables in programming causes logical errors that cost students time and effort while they try to resolve them. According to KAU programming instructors, student motivation to learn is negatively affected when such errors continuously appear with neither specific solutions nor compiler advice on solutions.

Research, as early as the 80’s, shows that such difficulty can be overcome by providing explicitly concrete models to students [14]. One of the concrete models implies the use of the Roles of Variables concept. ROVs generally characterize the dynamic nature of variables in various programs by the sequence of successive values a variable obtains and with no attention to the way the values are further used [9].

Empirical research related to ROV shows that only nine roles are needed to cover 99% of variables in novice level programs [11]. The formal definitions of these roles are:

- **Fixed value** (Constant): A variable whose value does not change after initialization.
- **Stepper**: A variable stepping through a systematic, predictable succession of values.
- **Follower**: A variable that always gets its new value from the old value of some other variable.
- **Most-recent holder**: A variable holding the latest value encountered in going through a succession of values, or simply the latest value obtained as input.
- **Most-wanted holder**: A variable holding the best, or otherwise most appropriate, value encountered so far.
- **Gatherer**: A variable accumulating the effect of individual values.
- **One-way flag**: A two-valued variable that cannot get its initial value once its value has been changed.
- **Organizer**: An array used for rearranging its elements.
- **Temporary**: A variable holding some value for a very short time only.

3. VISUALIZATION

Interactive visualization has been employed in computer science education since the 1980’s [5]. It has been used both in teaching algorithms [3, 4] and in teaching programming [7]. PlanAni, a program visualization tool, was created for the presentation and use of Roles of Variables in the classroom [12].

Existing visualization systems that deal with program variables can be categorized into two categories: Semi-automatic visualization systems that provide sets of ready-made representations for variables, and hand-crafted visualization systems that give more freedom for variable visualization [12].

3.1. Engagement Levels of Visualization

The “Taxonomy of an Engagement with Visualization” identified by the Working Group on “improving the educational impact of algorithm visualization” lists six different forms of learner engagement with visualization technology. These are: No viewing, Viewing, Responding, Changing, Constructing and Presenting [5].

Researchers have studied the effect of using only one level of student engagement with algorithm visualization [1, 3]. Furthermore, varying three levels of student engagement with algorithm visualization has been previously measured by Grissom et al [2], in which he found that increasing student engagement levels with algorithm visualization increases student performance.

3.2. PlanAni

PlanAni is a variable role-based program animation system, where each variable role has a stored visualization, or what is referred to as a role image. Role images give clues on how successive values of the variables relate to each other and to other variables. PlanAni also uses role-based animations of various operations like assignment and comparison. It is an easy to use program for novice students. However, animation commands must be authored manually for each program. Typically, five animation lines of Tool Command Language (TCL) code are required for each line of code in the original animated program.

4. EXPERIMENT METHODOLOGY

In this research, an experiment was conducted to test if visualizing the Roles of Variables enhances novice student learning in the CS102 course when the level of student engagement with visualization increases. Participating students were selected and assigned randomly to three different groups. Students in all three groups had the ROV concept explained to them in a traditional
An ROV group that had no interaction with the ROV visualization tool PlanAni.

A Viewing Visualization group that was only permitted to view the animated visualization of programs using PlanAni, which was run by the instructor.

A Responding to Visualization group in which students were allowed both to view the animated visualization of programs and to interact with the visualization tool PlanAni.

The research experiment lasted five weeks and covered control structures, iteration statements, strings, functions, parameter passing, arrays and structures are all included. The programming language used is C++. The course is accredited as four credit hours. However, students attend three hours of lecturing and a one and a half hour lab session per week.

The participating subjects were female undergraduate students from the Faculty of Science who were enrolled in the CS 102 course. The subjects were assigned randomly to one of the three experiment groups. In total, 91 students participated in the experiment as follows: 29 students were assigned to the ROV group, 30 students were assigned to the Viewing Visualization group, and 32 students were assigned to the Responding group.

The post-test exam is the first periodical exam of the course. It was used to collect data about student performance. Its complexity reflected the prevailing state of the course. It contained five questions addressing: problem representation, program simulation, debugging, program construction and a control question that didn't contain any variable usage.

Two questionnaires were also used: demographic and self-reporting questionnaires. The demographic questionnaires were presented at the beginning of the experiment and the self-reporting questionnaires were presented at the end. The self-reporting post-questionnaire, which the students filled out themselves, gave them the opportunity to assess their own programming performance in form of understanding levels of using and writing statements. The three groups of students attended the lecture hours together but were separated during the lab sessions. The ROV group received ordinary lab work and was asked to construct programs while focusing on variables roles; but didn't work at all with PlanAni. The Viewing Visualization group viewed PlanAni animations using the projector during the first fifteen minutes of the lab and spent the remaining time conducting traditional lab work as did the ROV group. The Responding to Visualization group used PlanAni on their PCs and were allowed to interact freely with the tool during the first fifteen minutes of the lab session while viewing the animations displayed by the projector. They also spent the remaining time conducting traditional lab work, as did the ROV group. The groups that used PlanAni completed all examples provided by the tool. At the end of the fifth week, all groups participated in the same post-test exam and filled out copies of the same questionnaires. This experiment procedure is summarized in figure 2.

5. RESULTS

All 91 students attended most of the sessions throughout the experiment duration and their results were used for this research. The students’ grades were transformed to a score using percentage equivalences. Table 1 lists the means of the Problem Representation score (REPRS), the Program Construction score (CONST), the Program Comprehension score (COMPR), the Program Simulation score (SIMUL), the Debugging score (DEBUG), the Control score (CNTRL), and the Exam Total grade (TOTAL).

| Table 1 Means for all Grades (out of 100) |
|-------------------|-------------------|-------------------|
| Groups            | ROV N=29          | Viewing N=30      | Responding N=32 |
| Scores            | Mean              | Mean              | Mean              |
| REPRS             | 66.09             | 72.78             | 65.62             |
| CONST             | 28.66             | 33.33             | 33.98             |
| COMPR             | 59.48             | 55.83             | 51.56             |
| SIMUL             | 14.37             | 14.44             | 9.38              |
| DEBUG             | 47.70             | 58.33             | 45.31             |
| CNTRL             | 61.38             | 48.67             | 60.63             |
| TOTAL             | 44.91             | 46.40             | 44.29             |

The student overall performance was calculated as the summation of the five separate scores. In general, the mean score for the Viewing group was the highest, followed by the ROV group, then the Responding to Visualization group. The means of the three groups were compared using an ANOVA test and the differences between the groups were not significant (F (2,88)=0.208, p=0.813).

The Control question was used to ensure that the differences in grades do not depend on the independent variable but reflect any variables that could not be controlled. This question was not related to variables in any way and it was in the form of constructing a little code segment that checks the password of an ATM card using a switch statement. The normality was not satisfied for this variable so the means were compared using a Kruskal-Wallis Test. The differences between the groups were not significant (Chi-square=1.7, df=2, p=0.427).

The Program Construction question required students to construct a program that finds the smallest digit and its position in any
integer number entered by the user, using an iteration statement. Since the normality was not satisfied, the means were compared using a Kruskal-Wallis Test. The differences between the groups were not significant (Chi-square=1.206, df=2, p= 0.547).

The Program Comprehension question measured the student ability to read program code and specify its functionality. The program code was 20 lines and it read a string and a character from the user and searched for the character in the string. If the character was found, it was deleted from the string and the new string was displayed; otherwise, a “not found” message was displayed. The means of the groups were tested using a Kruskal-Wallis Test because the normality was not satisfied. The differences between the groups were not significant (Chi-square=1.594, df=2, p= 0.451).

The Program Simulation question was a segment of code that depended on a nested loop to build a 4-layer Pascal triangle. The normality test of the scores was not satisfied and the Kruskal-Wallis test was used. The differences between the groups were not significant (Chi-square=0.747, df=2, p= 0.688).

The Debugging question evaluated the ability of the students to correct 4 compilation errors and 2 logical errors in a program that calculates the area of a triangle or a square, depending on a user’s choice. The program was about 30 lines of code. Again, normality was not satisfied, so a Kruskal-Wallis test was used. The differences between groups were significant (Chi-square=8.124, df=2, p= 0.017). Scores of the Viewing Visualization group were significantly higher than scores in the ROV group (U=296, p=0.028). In addition, scores of the students who used PlanAni to view the visualizations were significantly higher than scores of students who viewed and responded to the visualizations using PlanAni (U=300, p=0.009).

The Problem Representation question asked students to draw a flowchart to describe the procedure of a program that replaced each character c or C with a small c and converted all other letters in the string entered by the user to uppercase. The normality was not satisfied and the means were compared using a Kruskal-Wallis Test. The differences between the groups were not significant (Chi-square=3.610, df=2, p= 0.164).

6. DISCUSSION

The research shows insufficient evidence to conclude that students’ performance in programming is similar to when the levels of novice students’ engagement with visualization tools increase. This is due to the fact that statistical differences were not found in quantitative and quantified qualitative total programming scores.

Furthermore, the research shows insufficient evidence to conclude that students’ performance in problem representation, program construction, program simulation and program comprehension are similar when the levels of novice students’ engagement with visualization tools increase. This is due to the fact that statistical differences were not found in these scores at α=0.05. These results reflect the same statistical conclusion from the results presented by Sajaniemi and Kuittinen [13]. Although, by analyzing program summaries, this research found that performance in program construction and program comprehension is increased when the levels of novice students’ engagement with visualization tools increase.

However, the research shows that students’ performance in program debugging is different when the levels of novice students’ engagement with visualization tools increase. This is due to the fact that statistical differences were found in program debugging scores at α=0.05. Program debugging scores were improved when students’ engagement with visualization tools was limited to viewing visualization. Naps and Grissom [4] who compared no viewing, viewing and responding groups, found that the responding group exhibits significant improvement in a post-test compared to the no viewing group. This finding is somewhat reflected in the results of this research but the difference might have occurred because students in the Viewing group concentrated on the debugging process more than the Responding group who worked with the program and spent their time and concentration on entering data and responding to messages.

Through observation and monitoring of students and informal interviews and interaction with them both in spoken and written forms, about 50% of the students during lab sessions stated that they enjoyed working with PlanAni and they noted that it helped them understand program sequences and helped them better understand the Roles of Variables.

These findings are different from prior research, in which researchers found that the use of ROV visualization seems to foster the adoption of role knowledge and improve programming skills [13]. This could be due to the fact that previous research depended mainly on qualitative findings which heavily depended on student verbal feedback, satisfaction questionnaires, analyzing answers and observations, while this work included quantitative measures of understanding and learning such as test scores.

7. CONCLUSION

In this experiment, the PlanAni visualization tool was used. PlanAni was the only tool that supported the concept of Roles of Variables. While the findings show that debugging skills are improved by the use of PlanAni, there were limitations of PlanAni based on the pedagogical requirements of successful visualization tools [8].

To make PlanAni, or any other future visualization tool, a more pedagogical and powerful tool that is more useful in supporting the teaching of introductory and intermediate programming courses a few improvements are needed. The following is a list of suggestions:

- Adding needed features such as rewind capability and the capability to jump to specific animation states in either direction.
- The tool should support interactive prediction with stop and think questions. Without such questions, once a student becomes confused, continuing to watch visualization is similar to watching a movie in which one has lost interest.
- To avoid the state of using these questions as a guessing game, the system should allow the students to enter a “quiz for real” mode. In this mode, the student's responses to questions are recorded in a database on the server. Apart from providing valuable feedback for instructors, these results can be used in evaluating students.
- Some modifications to the interface are needed. Although there are different font types, sizes and styles, font is not comfortable for reading even after changing the font settings.
• The presentation of the role changes could be improved. The proper change may appear by displaying a new role image next to the old role image and transferring the data between two role images then making the old role image disappear, while the new one moves to its place. A sporadic change may appear by making the old role image with its data disappear then displaying an empty new role image.

• The new tool should be provided as a java applet or application to allow the widest possible target audience through on-line distribution.

The investigation of this research was applied to the computer programming education field. One of the implications of this study is to investigate the use of visualization in different contexts further. It may be applied in other courses in the Computer Science Department such as data structure or other fields of study. The study can also be expanded to study the effect of ROV on students’ different learning styles and vice versa.

The present study had certain limitations. First, the assignment procedure, limited by the fixed student schedules, their overlapping, and the inability to change them, resulted in the fact that the assignment had to be limited to those schedules. This limitation forced the researcher to group sections, which in turn resulted in larger class sizes, and sometimes less class control. The overall duration of the experiment was another limitation, as perhaps the five weeks were not long enough to ensure that all students understood all Roles of Variables and were familiar with them in PlanAni. In addition, lab periods were also limited in time. The limitation of the time could have affected the power of the methods and tools. However, the used times were the same actual times spent in a traditional CS 102 class. Finally, the major limitation in the measures was that no valid and reliable scale that measures programming understanding exists.

The conclusions, as well as the limitations of this study, bring forth some fruitful and interesting possible venues for future research. The first step would be to use other assessment tools to investigate programming performance. In fact, developing a tool for measuring programming understanding is a very important overlooked research area. The sample size of the study could be increased so the results become more significant. The experiment can be run again with a focus on Computer Science students only. Also, Extra lab time for visualization is needed and extra training time on visualization is also needed. Future work may expand the engagement levels to investigate all six levels. It would also be interesting to gain an understanding of the effect of these levels in relation to different types of courses.

Despite the limitations, the most important finding of this paper lies in the fact that viewing visualizations of ROV does indeed improve student debugging skills. This means that adopting such methods in programming courses could improve student learning by improving their debugging skills.

The researchers hope that however minute their findings are they contribute positively to the field of computer science education.

8. REFERENCES


