A STUDY OF ALICE: A VISUAL ENVIRONMENT FOR TEACHING OBJECT-ORIENTED PROGRAMMING

Jeraline Anniroot¹ and M.R. (Ruth) de Villiers²

¹Faculty of Accounting and Informatics DUT
²School of Computing UNISA

ABSTRACT

University students learning object-oriented programming (OOP) encounter many complexities. This paper describes a study in which the primary researcher undertook empirical research aimed at analysing learners’ interactions with the visual environment, Alice with rapid prototyping functionality. A questionnaire survey investigated the learners’ experience with the Alice environment and their understanding of OOP. Findings indicate that learners lack problem-solving abilities; are unable to grasp programming concepts on an abstract level; and spend insufficient time practicing programming exercises. Alice has proven to be an effective tool in helping to address some of these challenges and in improving learners’ grasp of OOP. Furthermore, the learners’ subsequent programming processes and performance were investigated. Results revealed that there was no statistically significant improvement in the performance of the learners who had been taught Alice in comparison to similar learners who were not exposed to the Alice intervention.

KEYWORDS

Alice, Object-oriented programming, Visual programming environments, Abstraction, Problem-solving, Motivation

1. INTRODUCTION

This study investigates the teaching and learning of computer programming, with the purpose of improving problem-solving skills and academic performance amongst second-year students of object-oriented programming (OOP) at the Durban University of Technology (DUT) in Kwa-Zulu Natal, South Africa. Novice programmers face various challenges and difficulties in learning OOP, in particular: demotivation; the complex syntax and semantics of an OOP language; the need for immediate feedback; difficulties in understanding compound logic and the application of algorithmic problem-solving skills.

The primary researcher is an Information Technology (IT) lecturer at the DUT, while the co-author is the supervisor of the former’s postgraduate studies. The primary researcher investigated the use of a 3D visual environment called Alice, with the aim of improving the understanding of fundamental programming concepts and imparting OOP skills. Alice is an open source teaching tool, designed to provide first-time exposure to learners on the basics of OOP. It allows learners to learn fundamental programming concepts whilst creating 3D animated movies and basic video games, thus providing an engaging interactive environment (Alice, 2010). The Alice system is a breakthrough in teaching OOP, one of its major strengths being that it can make abstract concepts concrete in the eyes of first-time programmers (Dann et al., 2009).

The study commenced during the first term of 2011 with cohorts of learners registered for the subject Development Software 2 (DS2) in the Department of Information Technology. A supplementary Alice workshop was held during lunch hours over a three-week period, where learners experienced hands-on interaction with the Alice software installed in the labs. The benefit of this participation was that learners could closely associate the Alice software with the corresponding course work in the DS2 syllabus. The sections on OOP were taught using Microsoft C#.Net and, for the experimental group, were supplemented with knowledge gained from also learning OOP in the Alice environment.
2. LITERATURE REVIEW

This section considers what other researchers have observed in relation to the teaching and learning of OOP. In addition, a brief overview is given of the Alice visual programming environment.

2.1 Teaching and learning object-oriented programming

Learning to program is considered hard for novices. Hence it is important to understand what makes learning how to program so difficult and how students learn (Matthews et al., 2009). Computer programming involves cognitive skills more than the acquisition of a body of knowledge (Hadjerrouit, 2008). It is a complex process, including understanding the task on hand, choosing appropriate methods, coding, testing and debugging an emerging program (Brooks, 1999).

Learners without prior programming experience are likely to be overwhelmed by the breadth and depth of material, thus contributing to attrition (Moskal, 2004). Developing good programming skills typically requires learners to do considerable intensive practice on programming exercises and to gain experience in debugging, which they cannot sustain unless they are adequately motivated (Law et al., 2009).

The teaching and learning of programming can be complex (Jenkins, 2001). Programming has evolved significantly from the traditional imperative (or procedural) programming languages and techniques to languages and techniques that emphasise object-oriented design and implementation (Phelps et al., 2005). Learning OOP involves writing programs in a language with a high level of complexity (Carlisle, 2009). The main reason why the OOP approach is difficult for most novice learners is because it is more abstract than the procedural approach (Hadjerrouit, 1999).

According to Sajaniemi (2008), students learning OOP experience problems not only in developing the required skills for writing programs, comprehension, and debugging, but also in understanding the basic object-oriented concepts. Object-orientation involves objects, classes, abstraction, encapsulation, inheritance, polymorphism, dynamic binding, and modularity, concepts that are used to understand problem situations, to design object-oriented models, and to choose appropriate means of implementation (Hadjerrouit, 1999).

This study highlights prominent challenges faced by learners of OOP, including the following:

(a) Lack of motivation for programming (Esteves et al., 2008; García-Mateos and Fernández-Aleman, 2009; Dann et al., 2009);
(b) Complex syntax and semantics (Winslow, 1996; Gomes and Mendes, 2007; Dann et al., 2009);
(c) Immediate feedback and identifying the results of computation as the created program runs (Wright and Cockburn, 2000; Galvez et al., 2009; Dann et al., 2009); and
(d) Difficulties in understanding compound logic and the application of algorithmic problem-solving skills (Gomes and Mendes, 2007; Esteves et al., 2008; Dann et al., 2009).

These challenges are addressed in Section 4.2.1, while Section 4.2.2 suggests possible solutions in the form of characteristics and methods that can support the teaching of OOP.

2.2 Alice

This study is founded on Alice, which is a rapid prototyping environment for 3D object behaviour, designed to help novice programmers develop interesting 3D animations (Zaccone et al., 2003; Dann et al., 2003; Kelleher and Pausch, 2006). Among the virtues of Alice as a learning tool are: concrete visualisation of objects and easy inheritance; ready motivation of interesting problems to solve; release from dealing with complex syntax mechanics; and immediate feedback, whereby logic errors become visually obvious (Johnsgard and McDonald, 2008; Dann et al., 2009).

These characteristics have contributed to the emergence of Alice over other visual programming environments that were developed to address challenges in teaching and learning programming, examples being Seymour Papert’s classic Logo (Folk, 1981), Karel the robot (Becker, 2001; Larason, 2005), Kara (Kiesmüller, 2009), BlueJ (Klölling and Rosenberg, 2001), Greenfoot (Montero et al., 2010), Second Life (SL) (Esteves, 2008) and MUPPETS (Phelps et al., 2005).
3. RESEARCH DESIGN AND METHODOLOGY

3.1 Research questions

The purpose of the investigation was to determine the effectiveness of implementing the Alice visual programming environment, with a view to improving the computer programming performance of second-level learners at the DUT. Guided by the questions following, the primary researcher conducted empirical research on the learners’ use of Alice and their subsequent programming processes and performance.

1. What is the effectiveness, as perceived by learners, of using the Alice visual programming environment and objects-first approach in addressing the challenges facing novice programming learners within the object-oriented domain?

2. To what extent do the test and exam results of participating learners relate to those of similar learners who were not exposed to the Alice intervention?

3.2 Research design

Every evaluation study has a research methodology and research criteria. Regarding the methodology, the study employed mixed-methods research, which according to Creswell and Clarke (2011), is a research design with philosophical assumptions as well as methods. As a method, it focuses on collecting, analysing and combining both quantitative and qualitative data in a single study or series of studies. This form of design helps to broaden understanding by incorporating both qualitative and quantitative research, or to use one approach to better understand, explain, or build on the results from the other (Creswell, 2009).

During the last week of the Alice workshop, learners were asked to complete a questionnaire regarding their experiences with, and opinions of, Alice. With regard to the evaluation criteria for this study, they were formulated from literature reviews of previous research. The data was processed by conducting quantitative analysis of the responses to closed-ended questions and qualitative analysis of responses to the open-ended questions. Concurrent collection of both quantitative and qualitative data, with the intention of comparing datasets to determine whether convergence occurs or differences emerge or if there is a combination of convergence and divergence, is known as the concurrent triangulation approach (Creswell, 2009).

Furthermore, a quantitative analysis was performed on the learners’ test and exam marks to compare the performances of learners who participated in the Alice workshop and similar learners who had been taught by conventional means.

3.3 Participants

The participants were second-year learners registered for DS2 at the DUT. The sampling strategy employed was non-probability sampling (Steyn et al., 1994), whereby there is a specific choice in whom or what is selected. The learners chosen to participate were selected from those who had passed all four first-year subjects at first attempt, which contributed to similarity between participants and ensured that they all had an average-to-good understanding of basic programming skills.

The experimental group initially comprised 50 learners. As the study progressed, attrition occurred in the lunch hour Alice workshop, and 21 respondents remained in the experimental group. Therefore, in the selection of learners for the comparison group, 21 with a similar success rate at first-year level were selected from the remaining learners, who had been taught OOP by conventional teaching methods only. Furthermore, the process implemented stratified sampling where each stratum was homogeneous with respect to certain characteristics (Steyn et al., 1994). For example, there was identical gender composition in each group.

3.4 Data collection and analysis

The questionnaire contained 25 closed-ended items, based on a 5-point Likert scale from 1 (Strongly Disagree) to 5 (Strongly Agree). Questions 1 to 10 investigated usability of the Alice visual programming environment and were therefore based on the interface design heuristics of Jakob Nielsen’s ten general principles (or criteria) for evaluation (Dix et al., 2004). An extensive usability study was undertaken but due
4. FINDINGS

This section presents the quantitative analysis of test marks and exam results, comparing performances of the experimental group and the comparison group. The section discusses responses to closed, quantitative questions from the questionnaire. The section also discusses responses to open, qualitative questions from the questionnaire. The questionnaire was completed by the experimental group only.

4.1 Quantitative analysis of test and exam results

The independent t-test was used to compare the mean values between the summative assessment performances of the two groups in their summative assessments. The results showed no significant differences between the mean values of the two groups. Although the results are not significantly different, mean scores in the experimental group were, in general, higher than the comparison group. Table 1 indicates that examination performance and the final mark were approximately 5% higher in the experimental group.

According to Clarke (1994), teaching methods delivered by many different media or by various mixtures of media, all produce similar learning results. Clarke’s claim appears to hold relevance in the present study. Similarly, Owusu et al. (2010) conducted a study with cohorts of learners from two randomly selected schools, exposing one group to computer-assisted instruction (CAI) and the other to conventional teaching methods. Results revealed that the CAI learners did not perform better than the conventional group. The study concluded that the use of CAI is not superior to the conventional approach. However, the learners in the CAI group found their e-learning exposure to be interesting.

Table 1. A comparison of the mean scores for the experimental group and the comparison group

<table>
<thead>
<tr>
<th>Group</th>
<th>DS101 (First-year results)</th>
<th>DS102</th>
<th>Test1</th>
<th>Test2</th>
<th>Exam Mark</th>
<th>Final Mark</th>
</tr>
</thead>
<tbody>
<tr>
<td>Comparison</td>
<td>63.0476</td>
<td>65.1905</td>
<td>64.7619</td>
<td>59.9524</td>
<td>49.2857</td>
<td>54.8095</td>
</tr>
<tr>
<td>Experimental</td>
<td>63.4286</td>
<td>69.2381</td>
<td>62.9524</td>
<td>69.9524</td>
<td>54.4762</td>
<td>59.1905</td>
</tr>
</tbody>
</table>

4.2 Quantitative analysis of closed-ended questions

4.2.1 Challenges faced by learners in learning object-oriented programming

Lack of motivation: While 90.5% of the experimental learners agreed that they are motivated to learn programming, only 66.7% admitted to spending a lot of time intensively practicing programming exercises, and a further 19% were unsure. Law, Lee and Yu (2009) state that in order to develop good programming skills, learners should do a lot of intensive practice on programming exercises to gain experience in debugging. This cannot be sustained unless they are adequately motivated. Further investigation is needed to establish the reasons behind the lack of practice done on the learners’ part.
Complex syntax and semantics: Winslow (1996), Kelleher and Pausch (2005), and Carlisle (2009) present a strong debate, affirming that learners frequently spend more time dealing with syntactical complexity and detailed issues of coding than on learning the underlying principles of object-orientation or solving the problem. A good percentage (61.9%) of the experimental participants agreed that learning the syntax and semantics of a programming language is challenging, while 28.6% were unsure. Only 52.4% agreed that it would be easier to learn how to solve a problem and learn basic concepts of object-orientation without having to deal with brackets, commas and semicolons, as is the case in Alice, while 38.1% were unsure. Further research conducted by Salim et al. (2010) indicates that rapid exposure to the terminology of programming syntax, such as do-while, if-then-else, switch-case, and for-next, may intimidate learners and result in poor performance, as well as learner attrition. Nevertheless, 55% indicated that they were not intimidated by direct exposure to programming syntax, such as do-while, if-then-else, switch-case, and for-next. Thirty five percent (35%) were unsure and only 10% agreed with the statement. Forty eight percent (47.6%) of participants in the experiment felt that the textual nature of the conventional programming environments used, makes it easy for them to learn how to program, while another 47.6% were unsure. Carlisle (2009) suggests that the textual nature of most programming environments works against typical learning styles, but the results of this study do not appear to support that. Although the learners enjoyed Alice, many of the experimental group were also comfortable learning OOP by conventional means.

Identifying results of computation as the program runs: Psychological evidence offered by Galvez, Guzman and Conejo (2009) suggests that feedback provided immediately after an error is the most effective pedagogical action. Sixty seven percent (66.7%) agreed that they were able to identify errors and correct them using the feedback given by the program. Seventy one percent (71.4%) felt that they were able to work independently on a program, from coding to testing, reaffirming the findings of Garcia-Mateos and Fernandez-Aleman (2009), which posit that current educational tendencies centre on the learners’ viewpoint rather than on the educators’, with the intention of creating independent, reflective and life-long learners. Due to experience from running and debugging programs, 76.2% of the learners felt that they were equipped to solve similar problems. A gulf of visualisation arises when programmer have difficulty mapping the observed behaviour of the running program against their mental model (Wright and Cockburn, 2000).

Difficulty of understanding compound logic: Research conducted by Gomes and Mendes (2007), Esteves et al. (2008) and Yu and Yang (2010) affirms that core problems experienced by many novice learners are the basic lack of problem-solving abilities and the difficulties experienced in using basic concepts, such as control structures, to create algorithms that solve concrete problems. The confidence levels of the experiment participants seemed quite high, in that over 76% claimed they were able to apply basic problem-solving techniques to create algorithms that solve problems; that they have a good understanding of pseudocode; and that they were able to decompose a large and complex programming task into smaller subtasks. Conversely, only 28.6% acknowledged using pseudocode to outline and understand the logic of a program before they started coding. One third (⅓) disagree with efforts to use pseudocode. There is a general inconsistency between the responses, which contradicts the learners’ claims that they do not experience difficulty in understanding compound logic.

4.2.2 How to improve the teaching of object-oriented programming

To address the challenges identified, this section proposes techniques to help teach OOP.

Algorithmic thinking and expression: Algorithmic thinking and expression involves the ability to read and write in a formal language (Dann et al., 2009). In the present study, 95.2% of the group agreed that they were able to read and write in a formal language. This claim will have to be investigated further.

Abstraction: According to Salim et al. (2010), many novice programming learners lack the ability to express their creative thinking in terms of programming abstractions, because they do not grasp programming concepts on an abstract level. This may lead to learners writing code without fully understanding every line. Although 76.2% agreed that they were able to use creative thinking and programming concepts to write programs, a third of the participants in the experimental group, acknowledged that on occasions they do write programs without understanding each piece of program code. With a further 19% of experimental learners being unsure about their ability to understand every line of code when writing a program, it can be safe to assume that, more often than not, learners are unable to grasp programming concepts on an abstract level.

Objects-first strategy: While 66.6% of the experimental learners agreed that they have a good level of understanding of objects, gained from their first year of study, 19% had disagreed. According to the ACM Computing Curricula 2001 (2010), the objects-first strategy commences by immediately introducing the
notions of objects and inheritance and then goes on to introduce more traditional control structures. However, only 47.6% of the learners felt confident that it would be easier to learn OOP during the first year of study, and later learn the conventional control structures such as loops, if statements etc., while a third disagreed. Phelps et al., (2005) state further that a way to minimise negative concerns and maximise the positive observations associated with object-oriented and objects-first programming, is to provide learners with a rich, interactive, collaborative virtual environment that supports the programming experience. A high percentage (85.7%) of the experimental learners stated that Alice helps them to see everything as an object.

3D animation authoring tools and visualisation: Several authors have stated that learners understand programming concepts better when given a visual representation. Three-dimensional animation assists in providing stronger object visualisation and a flexible, meaningful context for helping learners to perceive object-oriented concepts. Three-dimensionality provides a sense of reality for objects (Dann et al., 2003; Gomes and Mendes, 2007; Sajaniemi, 2008; and Carlisle, 2009). A very high percentage (90.5%) of the experimental learners agreed that a visual representation improves their understanding of programming concepts. Furthermore, 81% agreed that the visual effects in Alice provide a meaningful context for understanding classes, objects, methods, and events. Finally, 95.2% of the experimental learners agree that three-dimensionality makes objects seem real and that they were able to use Alice to write a new method to make objects perform animated tasks, such as hopping, flying, swimming etc.

4.3 Qualitative analysis of open-ended questions

Participants’ personal opinions and general attitudes emerged from the qualitative responses to the open-ended questions. All participants in the experiment expressed a liking for Alice. A pattern observed from 10% of experimental learners was their preference of having control over the program, while 10% enjoyed being able to exercise critical thinking skills. Participants appreciated not having to concern themselves with tedious typing. Others (10%) mentioned the enjoyment of interacting with other learners. Nearly 20% of the experimental learners found that Alice stimulates their creativity; 3D visual environments are more realistic than conventional programming software packages; and learners were motivated to improve their programming skills with time and practice.

In response to the effects of learning with Alice, a theme that emerged consistently (86% of participants) was acknowledgement that their understanding of OOP had improved. Moreover, 10% indicated that Alice enhanced their grasp of the basic programming concepts, such as selection and iteration (see Figure 4.3.1).

A pattern emerged that 67% of the participants felt that Alice’s visual programming environment helped them address some of the challenges they faced in learning OOP. Conversely, 10% doubted that Alice was useful in addressing these challenges. Nearly 62% of the experimental learners indicated that the visual feedback improved their understanding of programming concepts, corresponding with the findings of Gomes and Mendes (2007), Sajaniemi (2008), Carlisle (2009), and Montero et al., (2010), all of whom consider visualisation to be a mechanism that promotes the understanding of programming concepts. Approximately 30% of the learners expressed satisfaction with not having to concern themselves with syntax and semantics (see Figure 4.3.2).
5. CONCLUSION

The mixed-methods approach employed in this research involved quantitative and qualitative studies. This triangulated data collection and analysis, and contributed to confirming the findings. For example, in quantitative analysis of the closed-ended questions, 81% of experimental learners were found to agree that the visual effects in Alice provide meaningful contexts for understanding classes, objects, methods, and events. This corresponds with the general pattern observed from experimental learners in response to the qualitative open-ended Question 26.2 (see Figure 4.3.1), where 85.7% of learners believed that Alice had improved their understanding of OOP. Similar findings also emerged between the quantitative and qualitative studies, when 66.7% of experimental learners agreed in closed-ended responses that they could identify errors and correct them using the feedback given by the programs and, in the qualitative analysis of Question 26.5 (see Figure 4.3.2), 61.9% of the experimental group stated that the immediate feedback provided by Alice improves their understanding of programming concepts.

In response to the research questions in Section 3.1, Alice has, firstly, shown itself to be an effective tool that addresses challenges faced by novice programming learners within the object-oriented domain. Secondly, the findings in answer to the question on implementation of Alice in the classroom, do not, however, demonstrate a statistically significant improvement in learner performance when using Alice. The researchers recommend that a further study should be conducted in the 2012 academic year, with a greater and sustained number of participants and post-questionnaire interviews to strengthen the research.

REFERENCES


