Use of the *Alice* visual environment in teaching and learning object-oriented programming

Jeraline Dwarika
Faculty of Accounting & Informatics
Durban University of Technology
+ 27 84 6931380
annirootj@dut.ac.za and annirootj@gmail.com

M.R. (Ruth) de Villiers
School of Computing
UNISA
+ 27 12 3616080
dvillmr1@unisa.ac.za and ruth.devilliers1@gmail.com

ABSTRACT

Learners at tertiary institutions struggle with writing object-oriented programs in complex object-oriented programming (OOP) languages. This paper describes a study that sought to improve learners' understanding of programming in the domain of OOP. This was done through the use of a visual programming environment (VPE) called *Alice*, which was designed to help novice programmers learn OOP concepts, whilst creating animated movies and video games. A questionnaire was administered to obtain quantitative and qualitative data regarding learners' understanding of OOP and their experience with the *Alice* environment. Findings indicate that learners spend insufficient time on programming exercises and struggle with problem-solving, applying OOP concepts, and abstraction. However, the use of *Alice* addressed challenges faced by experiment participants within the object-oriented domain and improved their motivation to learn OOP. Further results revealed that the test and exam performance of learners who used *Alice*, was not statistically better than those of similar learners who were not exposed to the *Alice* intervention.

Categories and Subject Descriptors

D.1.5 [Programming Techniques]: Object-oriented Programming; D.3.3 [Programming Languages]: Language Constructs and Features – *abstract data types, classes and objects, inheritance, polymorphism, procedures, functions and subroutines*; I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism – *Animation*

Keywords

Abstraction, *Alice*, Motivation, Object-oriented programming, Problem-solving, Teaching and learning, Visualisation, Visual programming environments

1. INTRODUCTION

Concern over learner attrition, the lack of learner motivation and high failure rates in programming courses have generated a drive towards creative approaches to make undergraduate courses more attractive to learners and to contribute towards higher success rates [2].

Such concerns are prevalent in object-oriented programming worldwide [18, 29, 10], and also occur amongst students at the Department of Information Technology (IT) of the Durban University of Technology (DUT), South Africa, where the research was undertaken.

The purpose of the study was to determine the effectiveness of implementing the *Alice* visual programming environment (VPE), with a long-term view to improving the computer programming performance and learning experience of second-level IT learners. The intention was to expose them to new and exciting ways of programming that enhance problem-solving skills and nurture higher-order critical thinking. *Alice* is an open source teaching tool, designed to provide first-time exposure to learners on the basics of object-oriented programming (OOP). They can learn fundamental programming concepts whilst creating 3D animated movies and basic video games that contribute to an engaging interactive environment [6, 19].

In related work, studies have been conducted on the use of *Alice* with first-year university learners [6, 7, 20]. The present work differs, because it deals with learners who were exposed to OOP during their second year of study, having done only procedural programming in their first year. Previous work by the present researchers [2] investigated experiences of an earlier cohort who used *Alice* in a smaller-scale study where attrition occurred. The findings were useful, but there was insufficient learner feedback. That work served as a pilot study and learning curve for the major case study on which this paper focusses. In this study, the full group of participants was maintained to the end and the results are a new contribution.

In the cohort of learners registered for Development Software 2 (DS2) in the Department of IT, students were filtered based on their first-year results and the fact that they were doing OOP for the first time. The filtered students had the opportunity to volunteer for an experimental group that would participate in the study. A supplementary *Alice* workshop was held during lunch hours over a two to three-week period, where these participants experienced hands-on interaction with the *Alice* software installed in the labs and also did collaborative projects.

The performance of the experimental group was measured against that of a control group with a similar composition and academic history at first-year level. The learners in the control group were drawn from the other learners registered for DS2, who were taught OOP by conventional methods only. The comparison was done by analysing learner data from tests and examinations in both groups.

This paper overviews relevant literature (Section 2), explains the research design (Section 3) and reports selected findings (Section 4). Sections 5 and 6 respectively provide discussion and recommendations, while Section 7 concludes the study.

SAICSIT '15, September 28-30, 2015, Stellenbosch, South Africa
© 2015 ACM. ISBN 978-1-4503-3683-3/15/09$15.00
DOI: http://dx.doi.org/10.1145/2815782.2815815

2. LITERATURE REVIEW

This section considers related work by other researchers regarding the teaching and learning of OOP. In addition, a brief overview is given of the *Alice* visual programming environment.

2.1 Teaching and learning OOP

Learning to construct computer programs is considered hard for novices. Learners often struggle to develop the competencies and skills required to code programs that execute correctly. Hence, it is important to understand what makes learning how to program so difficult and how students learn [24]. It is considered difficult because "it requires learning about programming concepts and the language of programming at the same time" [19:3].

Furthermore, program execution is a dynamic process, and it is complex to mentally grasp and track how variables change during program execution. Learners have problems visualising all the changes that occur as a computer program runs. Programming involves understanding the task on hand, choosing appropriate methods, coding, debugging and testing an emerging program [4].

Furthermore, programming courses traditionally emphasise theoretical understanding of programming concepts, as well as application. The concepts are reinforced through practical hands-on experience. Learners without prior programming experience are likely to be overwhelmed by the breadth and depth of material, thus contributing to attrition [26].

One of the core challenges experienced by programming lecturers is developing and sustaining a high level of learner interest and motivation to learn programming. To develop good programming skills, learners are typically required to do considerable intensive practice on programming exercises and to gain experience in debugging, which they cannot sustain unless they are adequately motivated [22].

The teaching of programming, particularly OOP, is therefore as complex as learning how to program. Extensive research efforts have been invested in developing techniques to assist in teaching programming and learning programming. Programming has evolved considerably from the traditional imperative (procedural) programming languages and techniques.

This evolution has led to a greater emphasis on object-oriented design and implementation [28]. Learning OOP involves writing programs in a language with a high level of complexity [5]. Novice programmers tend to find the OOP approach difficult, mainly because it is more abstract than the procedural style [18].

According to [29], students learning OOP experience problems not only in developing the required skills for writing programs, understanding the relevant theory, and debugging, but also in grasping the underlying concepts of object-orientation. Object-orientation involves objects, classes, inheritance, encapsulation, polymorphism, abstraction, modularity and dynamic binding. These concepts are used to represent the problem situation, to design object-oriented models, and to decide on suitable means of implementation [18].

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

(a) Lack of motivation for programming [12, 15, 10];
(b) Complex syntax and semantics [32, 16, 10];
(c) Immediate feedback and identifying the results of computation as the created program runs [33, 14, 10]; and
(d) Difficulties in understanding compound logic and the application of algorithmic problem-solving skills [16, 12, 10].

2.2 *Alice 2.2*

Alice's innovative approach in teaching programming, aids educators in the instructional process and allows for easier assimilation by learners of traditional program-creating concepts. The authors of *Alice* 2.2 consider that Version 2.2 represents a breakthrough in teaching object-oriented computing. Objects in *Alice* are reified as 3D humans, furniture and animals, thereby making them easily visible, concrete and real.

Furthermore, the state of *Alice* objects can be changed by calling methods such as 'move forward one meter' or 'turn left a quarter turn'. Such object behaviours are intuitively and easily understood by learners. "One of *Alice*'s real strengths is that it has been able to make abstract concepts concrete in the eyes of first-time programmers" [10:11]. For example, Figure 1 depicts an animation 'Defending Naptime', that tells a story about a rabbit whose sleep is interrupted by a cell phone. This screenshot was taken during program execution, and presents the learner with a visual representation of the status of the program code. Learners are able to pause, play, restart and stop the animation, and toggle its speed. They can also take a picture at any point in time.

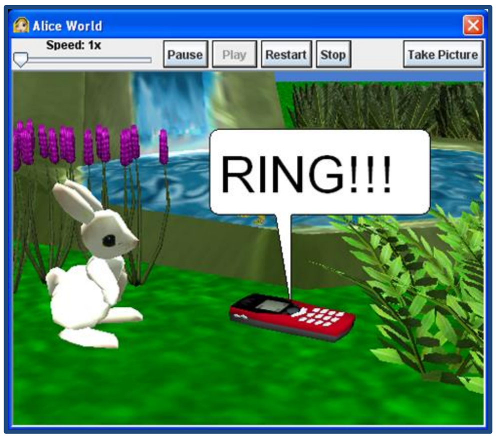
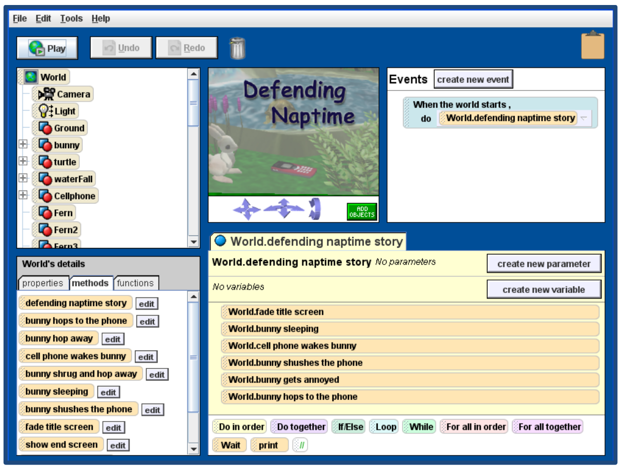


Figure 1. An initial scene in an *Alice* world during program execution, ©Dr W Dann



```
When the world starts ,
   do   World.defending naptime story
```

```
defending naptime story   edit
bunny hops to the phone   edit
bunny hop away   edit
cell phone wakes bunny   edit
bunny shrug and hop away   edit
bunny sleeping   edit
bunny shushes the phone   edit
fade title screen   edit
show end screen   edit
```

```
World.defending naptime story
World.defending naptime story  No parameters
No variables
    World.fade title screen
    World.bunny sleeping
    World.cell phone wakes bunny
    World.bunny shushes the phone
    World.bunny gets annoyed
    World.bunny hops to the phone
```

Figure 2. An *Alice* interface during the coding of an animation, ©Dr W Dann

Alice has an interactive interface, in which learners use drag-and-drop graphic tiles to formulate coding statements during program creation, as depicted in Figure 2. Learners are able to relate these instructions to standard statements in commonly used programming languages, such as C#, Java and C++. The VPE allows learners to immediately visualise the execution of their animation programs. Learners are thus easily able to understand the relationship between the programming statements and the behaviour of the animated objects. Moreover, they are encouraged to manipulate the objects in their virtual world, whilst gaining experience in programming concepts such as loops, if statements, properties, methods, functions, events, etc. *Alice* thus exposes the learner to the basic programming constructs typically addressed in introductory programming courses [6, 25, 31].

The features of *Alice* as a learning tool include the following:

(a) concrete visualisation of concepts such as objects and basic inheritance;
(b) motivation of learners, by providing interesting problems for them to solve;
(c) release from dealing with complex syntax mechanics, while errors in logic become visually obvious; and
(d) simplification of event-driven programming, which is interesting to explore in the *Alice* VPE [20].

These characteristics have contributed to the emergence of *Alice* as more popular than other visual programming environments developed to address challenges in teaching and learning programming, examples being Seymour Papert's classic *Logo* [13]; *Karel* the robot [3]; *Second Life* (*SL*) [12]; *MUPPETS* [28]; *Scratch* [23, 30] and *Lego Mindstorms* [21, 1].

3. RESEARCH DESIGN AND METHODOLOGY

3.1 Research questions

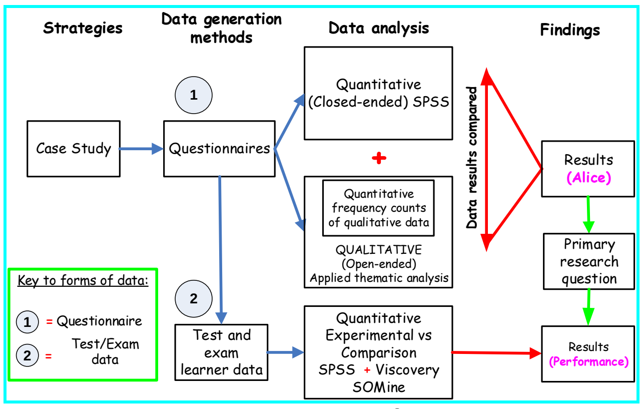
The aim of this research was to investigate the extent to which the implementation of the *Alice* visual programming environment in a second-level programming course at the Durban University of Technology could improve the performance and learning experience of learners. The research questions are:

1. *What is the effectiveness, as perceived by learners, of using the Alice visual programming environment 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

The research design of this study is based on Creswell's [2009] *Framework for Design*. The three vertices of this framework represent the philosophical worldview underlying a study, the selected strategies of enquiry, and the research methods used. The philosophical worldviews in this study are advocacy/participatory and pragmatic. A mixed-methods strategy of inquiry was employed, which according to Creswell and Plano Clark [2011], is a research design with philosophical assumptions as well as methods of inquiry. As a set of methods, it focuses on collecting, analysing and combining *quantitative* and *qualitative data*. Creswell [2009] [9] posits that this strategy of incorporating both qualitative and quantitative research, helps to broaden understanding, and also uses the one approach to better understand, explain, or build on the results from the other. The methods used in this study progress from the initial research questions through to data collection and analysis, followed by interpretation, write-up and validation.

Figure 3 represents the detailed research process followed in the case study. Questionnaires were administered to the participants of the *Alice* workshop. The quantitative questions measured their level of agreement or disagreement to the closed-ended questions in the questionnaire (Likert scale integer values, average rating, category rating).



Strategies
Data generation methods
Data analysis
Findings
Case Study
Questionnaires
1
Quantitative (Closed-ended) SPSS
+
Quantitative frequency counts of qualitative data
QUALITATIVE (Open-ended) Applied thematic analysis
Data results compared
Results (Alice)
Primary research question
Results (Performance)
Key to forms of data:
1 = Questionnaire
2 = Test/Exam data
2
Test and exam learner data
Quantitative Experimental vs Comparison SPSS + Viscovery SOMine

Figure 3. Research processes of the case study

Responses to the qualitative open-ended questions contributed towards eliciting rich, spontaneous findings and provided some interesting unanticipated findings. Triangulation of data was conducted between the quantitative findings (closed-ended questions) and the qualitative findings (open-ended questions), which led both to common findings and varying results. Qualitative interviews were also conducted but are excluded from this paper due to space constraints.

In further quantitative work, test and exam marks were used to compare the performances of the experimental group who participated in the *Alice* workshop with performances of the control group who did not participate in the workshop.

3.3 Participants

The participants were second-year learners registered for DS2, within the ND: IT programme at DUT. In order to vet candidates for attendance at the *Alice* workshop, criteria were established to filter potential participants according to subjects they had completed. A call for participation was made to the filtered candidates and a sample of volunteers responded. Fifty-five (55) were selected for the experimental group on a first-come-first-serve basis and they all signed informed consent.

The 50 other learners who qualified, formed the control group and were taught OOP by conventional teaching methods only, i.e. they did not participate in the *Alice* study. Permission was obtained from the institution to use their data. All the participants in both the experimental and control groups were doing OOP for the first time. We assumed that of the 55 in the experimental group, about 50 would complete the intervention. In fact, there was no attrition and all 55 participants remained to the end of the *Alice* workshop and completed the questionnaire.

3.4 Data collection and analysis

The first section of the questionnaire requested the participants' profiles and demographic details. This included student number, surname, first name(s), gender, age, race, email address, contact telephone number and class group. The second section contained 25 closed-ended items, based on a 5-point Likert scale from 1 (Strongly Disagree) to 5 (Strongly Agree). Olivier [2004] states that Likert scales are used by respondents to indicate the degree to which various statements apply to them. The questions related to varying aspects, namely:

Questions 1 to 10 investigated usability of the *Alice* VPE and were based on Jakob Nielsen's ten interface design heuristics for usability evaluation [11]. Questions 11 to 19 emerged from concepts in the literature and findings of previous studies on the teaching and learning of programming, the challenges faced by OOP learners, and ways of improving the teaching of OOP. Questions 20 to 25 were based on criteria identified by the researcher. They emerged from her personal ten-year involvement in teaching OOP to IT learners.

For the quantitative components, data analysis was performed using SPSS (Statistical Package for the Social Sciences). Further quantitative analysis on learner results in tests and examinations was performed using Viscovery SOMine. This is excluded from this paper due to space constraints.

The third section of the questionnaire was a composite question, Question 26, which had six open-ended subquestions, Questions 26.1 to 26.6. It elicited qualitative responses regarding the participants' experiences with the *Alice* environment, their consequent understanding of OOP, and improvements they would like to see in the teaching of OOP.

Qualitative data analysis involves the identification of patterns, relationships and themes. In this study, the qualitative data was analysed using applied thematic analysis (ATA) [17], which involved developing a codebook to quantify the qualitative responses to open-ended questions. "The ATA approach is a rigorous, yet inductive, set of procedures designed to identify and examine themes from textual data in a way that is transparent and credible" [17:15].

4. FINDINGS

The section discusses participants' responses to closed, quantitative questions in the questionnaire, as well as responses to the open, qualitative questions. The questionnaire was completed by the experimental group only. Quantitative analysis of test marks and exam results is also presented, comparing performances of the experimental group and the control group.

4.1 Quantitative analysis of closed-ended questions

4.1.1 How Alice addressed challenges faced by learners in learning object-oriented programming

Table 1 presents the percentage distribution of participants' responses to selected closed-ended questions on their experiences of learning OOP with *Alice*. Average values of the Likert ratings are given in the final column. Since (Strongly Agree) gave a rating of 5 and (Agree) gave a rating of 4, averages of 4 or more indicate high ratings.

The high scores for *match between the system and the real world* (means 4.75 and 4.36) show that learners appreciated how realistic 3D objects bring coding into reality, whilst reflecting real-world scenarios. Similarly, there were high ratings for *visibility of system status*, *user control and freedom*, and the *consistency and standards* of the *Alice* environment. In addressing *error diagnosis and prevention*, there were reasonably positive impressions regarding the level of error prevention offered by *Alice*, as 67% (combined A + SA) of the experimental participants agreed that the *Alice* software always gives error messages to prevent errors from occurring. Furthermore, 73% felt that the *Alice* interface does not cause the learner to make errors. There were high means for the criterion that *Alice* does not 'crash' (4.04) and 4.13 for quick and easy error recovery. With regard to *aesthetic and minimalist design*, 65.5% of the participants felt there was no irrelevant information in the *Alice* interface design that distracted learners and slowed them down. With an average rating of 3.62, it implied that that *Alice* had relatively appealing aesthetics.

Table 1. Percentage distribution of participants' responses regarding their experiences of *Alice*

Learners' experiences of *Alice*							
Criterion		**SD (%)**	**D (%)**	**N (%)**	**A (%)**	**SA (%)**	**Avg Rating**
Visibility of the system status	I am always aware of what is going on in the system.	0.0	1.8	1.8	43.6	52.7	4.47
	When I save a world in *Alice*, the system indicates that files are being saved.	0.0	1.8	3.6	20.0	74.5	4.67
Match between the system and the real world	The system uses words, terms and phrases that I can easily understand.	0.0	0.0	1.8	21.8	76.4	4.75
	The templates used for new worlds and the objects in the system gallery, relate to real-world objects that I encounter in my day-to-day experiences.	0.0	3.6	9.1	34.5	52.7	4.36
User control and freedom	I am comfortable with the level of control that I have over the system.	0.0	0.0	7.3	50.9	41.8	4.35
	Alice allows me the flexibility to use the environment to perform a task.	0.0	1.8	10.9	34.5	52.7	4.38
Consistency and standards	The *Alice* interface maintains a consistent look and feel.	0.0	0.0	9.1	58.2	32.7	4.24
	The startup dialog box, play button, main menu and tab controls are clearly and consistently displayed.	0.0	1.8	3.6	32.7	61.8	4.55
Recognition rather than recall	The actions to be taken and options available for selection are clear and visible at all times.	0.0	3.6	14.5	54.5	27.3	4.05
	I do not have to remember the information from a previous screen in order to proceed with the next one.	1.8	32.7	30.9	25.5	9.1	3.07
Flexibility and efficiency of use	*Alice* caters for beginner to expert users.	1.8	0.0	14.5	40.0	43.6	4.24

Criterion		SD (%)	D (%)	N (%)	A (%)	SA (%)	Avg Rating
Aesthetic and minimalist design	There is no irrelevant information in the *Alice* interface design that distracts me and slows me down.	1.8	16.4	16.4	49.1	16.4	3.62
Error diagnosis, recovery and prevention from errors	*Alice* does not crash while I'm using it.	1.8	5.5	20.0	32.7	40.0	4.04
	In cases where I encounter system errors, the system provides an appropriate error message in simple language.	1.8	7.3	21.8	43.6	25.5	3.84
	I can recover from mistakes quickly and easily.	1.9	3.7	13.0	42.6	38.9	4.13
	The *Alice* software always gives error messages to prevent errors from occurring.	0.0	5.5	27.3	34.5	32.7	3.95
Help and documentation	The four tutorials in the startup dialog box are useful in helping me to learn how to use *Alice*.	0.0	1.8	10.9	49.1	38.2	4.24
	The example worlds in the startup dialog box are useful.	0.0	0.0	12.7	52.7	34.5	4.22
Lack of motivation for programming	*Alice* has improved my motivation for programming.	1.8	1.8	9.1	54.5	32.7	4.15
Fragile mechanics of program creation, particularly syntax	It is easier to learn how to solve a problem and to learn the basic concepts of object-orientation without having to deal with brackets, commas and semicolons. (*Alice* shields learners from these distractions.)	3.6	12.7	14.5	29.1	40.0	3.89
Identifying results of computation as the program runs	*Alice* provides immediate feedback as the program runs.	0.0	0.0	20.0	50.9	29.1	4.09
Difficulty of understanding compound logic	*Alice* allows me to focus on problem-solving.	0.0	1.8	12.7	58.2	27.3	4.11
Appreciation of trial and error	When using *Alice*, I use trial and error to 'try out' individual animation instructions as I create new methods.	1.8	7.3	25.5	49.1	16.4	3.71
	I can visibly see the effect that each new animation instruction has on the animation.	1.8	12.7	30.9	50.9	3.6	3.42
Incremental construction approach	*Alice* has taught me how to program incrementally i.e. I write one method at a time, testing and running each piece.	0.0	0.0	14.5	49.1	36.4	4.22
Impact of *Alice* on understanding OOP concepts	Inheritance	0.0	3.6	23.6	41.8	30.9	4.00
	Methods	0.0	0.0	14.5	43.6	41.8	4.27
	Properties	0.0	0.0	18.2	47.3	34.5	4.16
	Functions	0.0	1.8	10.9	50.9	36.4	4.22
Impact of *Alice* on understanding basic programming concepts	Loops	0.0	0.0	20.0	38.2	41.8	4.22
	If..statements	0.0	3.6	18.2	40.0	38.2	4.13
	Data types	1.8	5.5	27.3	32.7	32.7	3.89
	Event-driven programming	1.8	1.8	25.5	41.8	29.1	3.95
Ability to collaborate	The *Alice* workshop has provided the opportunity to work in pairs with other learners and I have chosen to do so.	0.0	5.5	20.0	47.3	27.3	3.96
	The experience of working with other learners has helped me to learn the *Alice* programming environment.	0.0	7.3	20.0	52.7	20.0	3.85
Impact of *Alice* on DS2 learners	The *Alice* workshop relates directly to the sections on object-oriented programming covered in the Development Software 2 syllabus.	0.0	9.1	7.3	49.1	34.5	4.09
	I am interested in learning more about computer graphics and animation.	0.0	1.8	5.5	25.5	67.3	4.58
	I am interested in learning and working more with the *Alice* visual programming environment.	0.0	0.0	7.3	36.4	56.4	4.49
	I used *Alice* during my personal time after attending the first lesson of the *Alice* workshop.	1.8	10.9	16.4	41.8	29.1	3.85

It has been previously highlighted that novice programmers face various challenges and difficulties in learning OOP. These core issues include, but are not limited to:

Motivation for programming and *Problem solving*: Relating to these aspects, 87% (A + SA) of the experimental participants indicated that *Alice* had improved their motivation to learn programming, and 86% agreed or strongly agreed that the environment allowed them to focus on problem solving.

The findings that follow, are not all shown in Table 1. Sixty percent (60%) acknowledged spending a lot of time intensively practicing programming exercises, indicating an association between motivation and practice. This relates to the assertion made by [22], which states that, to develop good programming skills, learners should do a great deal of intensive practice to gain experience in debugging. However, the learner must be adequately motivated to sustain this level of competence. Further investigation may be required to establish reasons for the somewhat tentative responses to this question.

Complex syntax, logic and semantics: A fair percentage (47%) of the experimental participants agreed (A + SA) that it was challenging to learn the syntax and semantics of a programming language, while 35% were unsure. A good percentage, 69% (A + SA) believed that the feature of *Alice* whereby one does not have to deal with brackets, commas and semicolons, simplifies problem solving and learning the basic concepts of object-orientation. This demonstrated appreciation for *Alice*'s drag-and-drop feature, which releases learners from dealing with complex syntax.

However, 45% were not intimidated by direct exposure to programming syntax, and it was notable that 78% disagreed that the textual nature of conventional programming environments makes it difficult to learn how to program. Carlisle [2009] [5] suggests that the textual nature of most programming environments works against typical learning styles, but the present results do not appear to support this assertion. Seventy-eight percent were comfortable learning OOP by conventional means, as well as enjoying the visual experience with *Alice*.

Immediate feedback and identifying results of computation as a created program runs: The speed of feedback was rated with a mean of 4.09, while 80% (A + SA) felt that the feedback was immediate. Seventy-six percent could identify errors and correct them using the feedback given by a program. Ninety-three percent were able to work independently on a program, from coding through to testing. Due to their prior experience of running and debugging programs, 89% of the participants felt that they were equipped to solve similar problems in OOP.

Difficulties in understanding compound logic and the application of algorithmic problem-solving skills: The confidence levels of participants in the experiment were fairly high, in that 78% claimed they were able to apply basic problem-solving techniques to create algorithms. However, it is of interest and somewhat of a paradox that, although 69% felt they had a good understanding of pseudocode and 66% stated they were able to decompose a large, complex programming task into smaller subtasks, only 35% had actually used pseudocode to outline and understand the logic of a program before they started coding. Furthermore, a third (33%) of the participants disagreed with using pseudocode to help in understanding the logic. This appears to contradict the participants' claims that they do not experience difficulty in understanding pseudocode.

Returning to items included in Table 1 and referring to the four core concepts, *inheritance, methods, properties and functions*, that form the foundations of learning OOP, 72%, 85%, 82% and 87% of participants, respectively, agreed (A + SA) that *Alice* had helped them to *understand*. Furthermore, percentages ranging between 65% and 80% agreed that *Alice* helped to improve their *understanding of iteration, selection, data types and event-driven programming*.

A high percentage (84% of participants) found that the *Alice* workshop *related directly to OOP concepts* covered in the DS2 syllabus. A great majority (93%) expressed interest in learning more about computer *graphics and animation*, with 93% also *eager to work more* with *Alice*, and 100% *wanting to learn more* about OOP. Seventy-one percent had used *Alice* during their *personal time* since the workshop intervention had commenced. These positive experiences encourage future use of *Alice* in teaching and learning OOP.

4.1.2 How to improve the teaching of object-oriented programming

To address the challenges, this section suggests techniques to help in teaching OOP. Some of the points here are not included in Table 1, although they emanate from the closed questions.

Objects-first strategy: While 69% of the participants in the experiment felt they had a sound understanding of objects, gained from their first year of study, 7% had disagreed. A fair percentage (58%) 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 27% disagreed. A high percentage (91%) stated that *Alice* helped them to view everything as an object.

3D animation authoring tools and visualisation: Participants' responses to this criterion were positively influenced by their exposure to *Alice*. Eighty five percent agreed that a visual representation improved their understanding of programming concepts. Moreover, 95% agreed that *Alice's* visual effects provided a meaningful context for understanding classes, objects, methods, and events and that they could use the *Alice* environment to write new methods to make objects perform animated tasks. Finally, 91% agreed that three-dimensionality made objects seem real.

4.2 Qualitative analysis of open-ended questions

The same two points that were addressed in the quantitative section, are now considered for the qualitative responses.

4.2.1 How Alice addressed challenges faced by learners in learning object-oriented programming

In responding to the open-ended questions, which addressed similar territory to the closed questions, participants provided spontaneous unprompted feedback about some of the challenges in learning OOP:

A notable 25% expressed difficulties, and even inabilities, in solving problems and applying programming concepts. Furthermore, 20% claimed to have a poor understanding of instantiation (i.e. creating an instance of an object). This is unsatisfactory, particularly in view of the fact that learners had practiced instantiation since their first year at DUT. Thirteen percent of the experiment participants spontaneously admitted that they had experienced difficulty in understanding the logic of methods. Moreover, a theme emerged from 11% who had had difficulties with inheritance. Other challenges were the need to remember syntax, the inability to understand every line of code, problems in debugging, and so on.

Table 2 provides some unprompted responses given by the participants regarding their experiences in using *Alice*. The responses are paraphrased into themes. The text that follows, discusses certain rows in the table.

Regarding the *match between the system and the real world*, 24% spontaneously described how they enjoyed using *Alice*'s graphics and animation. Regarding *aesthetic and minimalist design*, 29% explicitly stated that *Alice*'s interface is easy to use. Thirty-five percent reported that *Alice* enhanced their grasp of OOP concepts, such as methods, functions, events, inheritance, properties, parameters, classes, objects, instantiation and polymorphism. Thirteen percent attributed this improvement to the visual nature of *Alice*.

Thirteen (24%) felt that programming through visualisation alleviates the learner from having to remember syntax and code. Eleven percent appreciated seeing the effects of every statement of code, i.e. immediate feedback.

Furthermore, eleven percent found the *Alice* environment to be engaging and fun, whilst stimulating a greater interest in programming.

Table 2. Spontaneous responses on learners' experiences when using *Alice*

Criterion		**Frequency counts**	**%**
Match between the system and the real world	Graphics and animation make things more real	13	24
	Dealing with concrete objects associates the code with real-world objects	6	11
User control and freedom	Learning how to create movies/storytelling	2	4
	Learning how to develop video games	2	4
Recognition rather than recall	Drag-and-drop feature limits typing	9	16
	Alice releases learners to focus on problem-solving	2	4
	No complex syntax, only English-like statements	2	4
Flexibility and efficiency of use	User-defined methods are used to manipulate objects and can be tested individually	4	7
Aesthetic and minimalist design	The *Alice* interface is simple and easy to use	16	29
Working with *Alice* has improved my understanding of OOP	I have a better understanding of OOP concepts such as methods, functions, events, inheritance, properties, parameters, classes, objects, instantiation and polymorphism	19	35
	I can create methods to manipulate and animate objects to perform actions	6	11
	Everything in *Alice* is viewed as an object	6	11
Alice as a VPE can help address challenges faced by learners in learning object-oriented programming	It is easier to learn programming through visualisation and graphics, than having to remember the syntax for coding, e.g. I can see creation of methods, see objects move on command, and see the visual effects of every statement of code	13	24
	Alice is fun, engaging and cultivates an interest in programming	6	11
	An interactive environment that represents real-life situations	5	9
	It makes programming concepts easy to learn and understand	24	44

4.2.2 Spontaneous responses on how the teaching of OOP could be improved

In order to address the challenges identified in Section 4.2.1, the participants were asked to suggest techniques that would help alleviate the issues.

Following their positive experiences in using the *Alice* VPE, 27% suggested that a visual, graphical environment, such as *Alice*, should replace or supplement the conventional tools used in teaching OOP. Participants believed this would improve their interest and motivation to learn OOP.

A response from 25% of the experiment participants, requested that lecturers should explain programs in more detail, supplemented with practical examples to concretise theoretical concepts. Participants were not keen on merely being given solutions to problems without associated discussions. Eighteen percent (18%) requested that the pace of lecturing be slowed down to afford learners more time to grasp new concepts.

4.3 Quantitative analysis of test and exam results

Inferential statistical analysis was applied to the final marks to compare the performance of learners from the experimental group with those from the control group. For the data sets that were normally distributed according to the Shapiro-Wilk Test ($p > 0.05$), parametric tests (2-tail t–test) were used to determine significance of differences. For the others, non-parametric tests (Mann-Whitney) were used. The result of the t-test for the comparison of the means is given below:

Null hypothesis: difference = 0

t statistic = 1.143

Two-sided p-value = 0.256

The null hypothesis claims that there is no difference in the mean values between the two groups. The traditional approach to reporting a result (of a hypothesis test) requires a statement of statistical significance. A p-value is generated from a test statistic. A significant result is indicated with "$p < 0.05$". In this study, NONE of the p-values were significant. This means that statistically, there are no significant differences between the average scores of the experimental group and the control group.

Although the results are not significantly different, the last two rows of Table 3 show that the mean examination mark of the experimental group was 2.6% higher than that of the control group and the final mark was 2.8% higher. The authors acknowledge, however, that there is no evidence that the difference in the sample means would be reflected in the population means. Some attrition occurred in the control group, where only 48 of the 50 wrote the exam.

Table 3. t-test for equality of means, mean scores and standard deviation for the experimental group and the control group

Assessment	Test	Sig. (2-tailed)	Group	N	Mean	Std. Devia-tion
DS101	Mann Whitney U = 1190.00	0.943	Control	48	67.4	12.7
			Experim	50	67.3	11.4
DS102	Mann Whitney U = 1005.50	0.166	Control	48	66.4	12.0
			Experim	50	69.7	8.6
Test	t-value = -0.509	0.612	Control	49	59.5	16.0
			Experim	55	61.0	14.4
Exam Mark	Mann Whitney U = 1189.50	0.388	Control	48	71.3	16.1
			Experim	55	73.9	14.5
Final Mark	t-value = 1.143	0.256	Control	48	66.1	12.8
			Experim	55	68.9	11.3

According to Clarke [1994] [8], teaching methods delivered by different media or by combinations of media, tend to produce similar learning results. Similarly, Owusu, Monney, Appiah and Wilmot [2010] [27] investigated cohorts of learners from two different schools, where one group was exposed to computer-assisted instruction (CAI) and the other to conventional teaching. Results revealed that the CAI learners did not perform better than the conventional group, indicating that the use of CAI was not superior to the traditional approach. However, the learners in the CAI group found their e-learning exposure interesting. These claims are relevant to the present study, where although no significant difference occurred, participants found that the *Alice* environment added value to their learning of OOP concepts. As stated in Section 3.2, interviews were conducted, but are not reported in the present paper for reasons of space. In brief, however, the unprompted data from 18 interviewees included spontaneous praise regarding the use of *Alice* for learners starting OOP. *Alice* supported understanding, self-learning and collaboration in ways that were fun, enjoyable and interesting.

5. DISCUSSION

This paper presents selected empirical findings which emanated from a case study at DUT. The mixed-methods approach of Creswell [2009] [9] involved quantitative and qualitative studies, which triangulated data collection and analysis, and strengthened the findings, although the researchers acknowledge that qualitative data can be subject to bias. Commonalities arose where similar findings occurred across two or three methods, thus confirming the findings. Contrasting results also occurred, showing the complementary value of mixed methods.

Examples of common findings:

Although percentages in the qualitative study are lower than those in the quantitative, it is emphasised that they came from unprompted responses, unlike the quantitative responses, where participants selected from options.

In response to the open-ended qualitative questions, eleven participants spontaneously indicated that they had a poor understanding of creating and instantiating objects. Similarly, in the closed-ended questions, a good percentage (69%) found that learning the basic concepts of OOP was easier when they were relieved by *Alice* from dealing with complex syntax. This confirmed the difficulties they experienced in applying basic concepts, such as creating and instantiating objects.

Another example emerged when the responses to closed-ended questions showed participants' desire to formally learn OOP via a visual, graphical environment such as *Alice*. They felt that this would increase their interest in OOP and improve the motivation to learn it. Similarly, by means of qualitative feedback, fifteen such requests (27%) came from responses to the open-ended questions. Further common findings emerged from the quantitative study, with a combined agreement of 91% in favour of using 3D visual tools to improve understanding of OOP.

Example of a contrasting finding

There were contrasting findings regarding techniques for improving the teaching of OOP, showing that individual learning styles differ. Open-ended questionnaire responses showed that some participants required in-depth support and liked being guided through the program logic with step-by-step instructions. Conversely, 40% of the closed-ended questionnaire responses indicated that the respondent could independently write a program, understanding each line of code. Although these statistics are not significantly high, they are sufficient to show the value of the unanticipated data that emerges from qualitative research.

6. RECOMMENDATIONS AND FUTURE RESEARCH

Based on the results of this study, the researchers propose:

(a) The positive results presented in this paper motivate the incorporation of *Alice* as a component of programmes for teaching and learning OOP.

(b) Human beings are known to learn from pictures. Gomes and Mendes [2007] [16] point out that many tools exist for solving programming complexities by means of graphical, animation and simulation techniques, capitalising on the potential of the human visual system. The participants responded enthusiastically and positively to visualisation. It is recommended that, no matter which visual programming environment is implemented, at least one such intervention should be used in tertiary institutions to supplement the teaching and learning of OOP.

(c) Robotic software, and in particular the *Lego Mindstorm NXT* robot, can provide an affordable, flexible and fun learning experience for programming learners. This could contribute to solving two of the challenges, namely, *lack of motivation for programming* and *the need for immediate feedback and response*.

(d) The *Alice* intervention provided a platform for learners to collaborate with each other, whilst learning in an engaging environment. It is recommended that other support structures, such as formal peer-to-peer negotiated programming and paired-programming, be adopted to foster collaborative learning. For example, the informal system in place at DUT, whereby third-year learners assist their second-year counterparts, could become a constructive and rewarding formalised initiative. It is widely believed that an excellent way to learn and to consolidate learning, is to teach. In so doing, senior learners could solidify their own prior knowledge. This could be implemented as early as the first year of study.

(e) It is further recommended that *Alice* workshops be conducted with first-year computing learners. This would provide an interesting learning experience, along with a gentle introduction to basic programming concepts.

(f) Learners experienced difficulties in problem solving. Logical programming is developed through sound pre-code planning and organisation, assisted by tools such as flowcharts, pseudocode and algorithms. The introduction of an additional subject on 'Logic' into Computing and IT programmes would benefit learners by improving their motivation and confidence to write programs.

(g) Tertiary institutions should continuously strive to ensure quality education. International and national best practice recommendations around quality dictate that professionals, such as doctors and engineers, attend continuing professional development (CPD). Formal compulsory intervention on state-of-the art technologies is of equal worth for Computing academics, who impart market-oriented knowledge and support the transfer of skills. Incentives should be in place to encourage lecturers to take training course, particularly discipline-based, structured CPD programmes, including courses on visualisation.

(h) A further study should be conducted using the latest version, *Alice* 3, in comparison to the set of related tools presented in the last paragraph of the literature overview. This version contains more explicit support for transitioning to Java.

7. CONCLUSION

The mixed-methods approach, combining quantitative and qualitative research, broadened understanding as it triangulated the findings. This section briefly revisits the research questions in Section 3.1:

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

In response to this, *Alice* has shown itself to be an effective tool that explicitly addresses challenges faced by programming learners in the object-oriented domain. The learning experiences with *Alice*, as described in Sections 4.1 and 4.2, motivated the participants and improved their problem-solving skills, as well as facilitating the construction of programs.

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?*

The findings in Section 4.3 demonstrate that the performance in object-oriented programming by learners in the experimental group was not significant when compared with that of learners in the control group. Nevertheless, it does provide a high quality learning experience, as attested by participants' feedback that *Alice* improved their enjoyment and understanding of programming concepts.

8. ACKNOWLEDGEMENTS

The authors expressed gratitude to the *Alice* team for permission to use screen shots of the software.

9. REFERENCES

[1] AGARWAL, R., HARRINGTON, D. & GUSMAN, C. 2012. Lego Mindstorm NXT controller with peer-to-peer video streaming in Android. *Journal of Computing Sciences in Colleges,* 27, 243-252.

[2] ANNIROOT, J. (now DWARIKA, J.) & de Villiers, M.R. A study of *Alice*: A visual environment for teaching object-oriented programming. Proceedings of the IADIS International Conference Information Systems 2012, 10-12 March 2012 Berlin, Germany. IADIS Press.

[3] BECKER, B. W. Teaching CS1 with Karel the robot in Java. SIGCSE '01: Proceedings of the thirty-second SIGCSE technical symposium on Computer Science Education, 2001 Charlotte, NC, USA. ACM, 50-54.

[4] BROOKS, R. 1999. Towards a theory of the cognitive processes in computer programming. *International Journal of Human-Computer Studies,* 51, 197-211.

[5] CARLISLE, M. C. 2009. Raptor: a visual programming environment for teaching object-oriented programming. *Journal of Computing Sciences in Colleges,* 24, 275-281.

[6] CARNEGIE MELLON UNIVERSITY. 2012. *Alice: an educational software that teaches students computer programming in a 3D environment* [Online]. Carnegie Mellon Foundation. Available: http://www.alice.org/index.php?page=what_is_alice/what_is_alice [Accessed 08 August 2010].

[7] CHANG, C., LIN, Y.-L. & CHANG, C.-K. 2013. Using Visual Programming Language for Remedial Instruction: Comparison of *Alice* and *Scratch*. In: WANG, J.-F. & LAU, R. (eds.) Advances in Web-Based Learning – ICWL 2013. Springer Berlin Heidelberg.

[8] CLARK, R.E., 1994. Media will Never Influence Learning. Educational Technology Research and Development, Vol.42, No. 2, pp. 21–29.

[9] CRESWELL, J.W., 2009. *Research Design. Qualitative, quantitative and mixed methods approaches.* Third Edition. Sage Publications, United States of America.

[10] DANN, W. P., COOPER, S. & PAUSCH, R. 2009. *Learning to program with Alice,* Upper Saddle River, New Jersey, Pearson Education, Inc.

[11] DIX, A., FINLAY, J., ABOWD, G. D. & BEALE, R. 2004. *Human-computer interaction,* Harlow, England, Pearson Education Limited.

[12] ESTEVES, M., FONSECA, B., MORGADO, L. & MARTINS, P. Contextualization of programming learning: A virtual environment study. FIE '08: Proceedings of the 38th Annual Frontiers in Education Conference, 22-25 October 2008 Saratoga Springs, NY. IEEE, F2A-17-F2A-22.

[13] FOLK, M. 1981. Review of "Mindstorms: Children, Computers, and Powerful Ideas by Seymour Papert", Basic Books: New York, 1980. *ACM SIGCUE Outlook,* 15, 23-24.

[14] GÁLVEZ, J., GUZMÁN, E. & CONEJO, R. 2009. A blended E-learning experience in a course of object oriented programming fundamentals. *Knowledge-Based Systems,* 22, 279-286.

[15] GÁRCIA-MATEOS, G. & FERNÁNDEZ-ALEMÁN, J. L. A course on algorithms and data structures using on-line judging. ITiCSE '09: Proceedings of the 14th annual ACM SIGCSE conference on Innovation and technology in computer science education, July 2009 Paris, France. ACM, 45-49.

[16] GOMES, A. & MENDES, A. J. An environment to improve programming education. CompSysTech '07: Proceedings of the 2007 international conference on Computer systems and technologies, June 2007. ACM, IV.19-1-IV.19-6.

[17] GUEST, G., MACQUEEN, K. M. & NAMEY, E. E. 2012. *Applied thematic analysis,* Thousand Oaks, California, Sage Publications, Inc.

[18] HADJERROUIT, S. A constructivist approach to object-oriented design and programming. ITiCSE '99: Proceedings of the 4th annual SIGCSE/SIGCUE ITiCSE conference on Innovation and technology in computer science education, June 1999 Cracow, Poland. ACM, 171-174.

[19] HERBERT, C. W. 2011. *An introduction to programming using Alice 2.2,* United States of America, Course Technology, Cengage Learning.

[20] JOHNSGARD, K. & MCDONALD, J. Using *Alice* in overview courses to improve success rates in Programming 1. CSEET '08: Proceedings of the 21st Conference on Software Engineering Education and Training, 14-17 April 2008. IEEE, 129-136.

[21] KLASSNER, F. A case study of Lego Mindstorms suitability for artificial intelligence and robotics courses at the college level. SIGCSE '02: Proceedings of the 33rd SIGCSE technical symposium on Computer science education, 27 February - 3 March 2002 Covington, Kentucky, USA. ACM, 8-12.

[22] LAW, K. M. Y., LEE, V. C. S. & YU, Y. T. 2010. Learning motivation in e-learning facilitated computer programming courses. *Computers & Education,* 55, 218-228.

[23] MALONEY, J., BURD, L., KAFAI, Y., RUSK, N., SILVERMAN, B. & RESNICK, M. Scratch: A sneak preview. C5 '04: Proceedings of the Second International Conference on Creating, Connecting and Collaborating through Computing, 29-30 January 2004. IEEE, 104-109.

[24] MATTHEWS, R., HIN, H. S. & CHOO, K. A. Multimedia learning object to build cognitive understanding in learning introductory programming. MoMM '09: Proceedings of the 7th International Conference on Advances in Mobile Computing and Multimedia, 14-16 December 2009 Kuala Lumpur, Malaysia. ACM, 396-400.

[25] MONTEIRO, I., DE SOUZA, C. & TOLMASQUIM, E. 2015. My Program, My World: Insights from 1st-Person Reflective Programming in EUD Education. In: DÍAZ, P., PIPEK, V., ARDITO, C., JENSEN, C., AEDO, I. & BODEN, A. (eds.) End-User Development. Springer International Publishing.

[26] MOSKAL, B., LURIE, D. & COOPER, S. Evaluating the effectiveness of a new instructional approach. SIGCSE '04: Proceedings of the 35th SIGCSE technical symposium on Computer science education, 3-7 March 2004 Norfolk, Virginia, USA. ACM, 75-79.

[27] OWUSU, K. A., MONNEY, K. A., APPIAH, J. Y. & WILMOT, E. M. 2010. Effects of computer-assisted instruction on performance of senior high school biology students in Ghana. *Computers & Education,* 55, 904-910.

[28] PHELPS, A. M., EGERT, C. A. & BIERRE, K. J. MUPPETS: Multi-user programming pedagogy for enhancing traditional study: An environment for both upper and lower division students. FIE '05: Proceedings of the 35th Annual Conference Frontiers in Education, 19–22 October 2005 Indianapolis, IN. IEEE, S2H-8-S2H-15.

[29] SAJANIEMI, J., KUITTINEN, M. & TIKANSALO, T. 2008. A study of the development of students' visualizations of program state during an elementary object-oriented programming course. *ACM Journal on Educational Resources in Computing (JERIC),* 7, 3:1-3:31.

[30] SANDOVAL-REYES, S., GALICIA-GALICIA, P. & GUTIERREZ-SANCHEZ, I. Visual learning environments for computer programming. CERMA '11: Proceedings of the Electronics, Robotics and Automotive Mechanics Conference, 15-18 November 2011. IEEE, 439-444.

[31] SHANNON, L.-J. & WARD, Y. 2014. A Case Study: From Game Programming to ICTs. In: KAUR, H. & TAO, X. (eds.) ICTs and the Millennium Development Goals. Springer US.

[32] WINSLOW, L. E. 1996. Programming pedagogy-a psychological overview. *ACM SIGCSE Bulletin,* 28, 17-22.

[33] WRIGHT, T. & COCKBURN, A. Writing, reading, watching: A task-based analysis and review of learners' programming environments. IWALT '00: Proceedings of the International Workshop on Advanced Learning Technologies, 2000. IEEE, 167-170.