International Journal of Engineering Education Vol. 27, No. 5, pp. 1027–1036, 2011
Printed in Great Britain

0949-149X/91 $3.00+0.00

Comparison of Two Approaches when Teaching Object-Orientated Programming to Novices*

DESMOND ADAIR
School of Engineering, Private Bag 65, University of Tasmania, TAS 7001, Australia. E-mail: dadair@utas.edu.au

MARTIN JAEGER
School of Engineering, Private Bag 65, University of Tasmania, TAS 7001, Australia

JONATHAN STEGEN
Engineering Department, Australian College of Kuwait, PO Box 1411, Safat, 13015, Kuwait

It has been stated several times in the literature that novice students must grasp object-orientated concepts immediately as the fundamental knowledge for programming using Java. Also, that introducing students to programming using the simpler procedural concepts early only compounds the difficulty of teaching object-orientated programming, as the need to always use some aspect of object-based code in Java cannot be avoided. Attempting to disguise this eventually causes frustration and confusion, even for good students. This paper presents the results of a comparison that evaluates, using a pre-test–post-test control group design, two approaches to teaching Java, where one approach uses objects first and the other uses a procedural followed by an objects approach. The results of the empirical study indicate that the students, who were first year engineers, using the objects first approach do indeed gain a better understanding of programming. This finding is supported by information gathered from a debriefing questionnaire, where the objects first approach was rated as easier for acquiring Java programming knowledge and skills.

Keywords: object-orientated programming; learning effectiveness

1. Introduction

There is a general consensus in the literature that learning to program is not easy [1–3]. Programming courses are perceived by many introductory students as being difficult [4] and indeed this is testified to by high drop-out rates and the fact that some students are still unable to write meaningful programs after the course [5].

Surveys by questionnaires have already been carried out to elicit the factors that are related to the difficulties of learning to program using objected-orientated languages and a number of papers have been written that address some of the problems that often occur when teaching the Java programming language [6–9]. Object-orientated concepts have been identified as the fundamental knowledge that students must have in order to perform well [9–12]. Importantly, they state that introducing students to programming using the simpler procedural concepts only leads to unnecessary difficulties, as the need to always use some aspect of object-based code in Java cannot be avoided, and that attempting to disguise this eventually causes frustration and confusion, even for good students.

While it has been stated many times in the literature that novice students must grasp object-orientated concepts immediately for efficient acquisition of programming skills, there actually has been very little objective testing using a controlled experiment to test this view. Some examples can be found in [13–15]. There is much anecdotal evidence and that obtained by surveys [2, 3, 5, 6, 12]. This however is subjective in nature and open to many influences that are not easily controlled. The contribution here is to provide evidence either way by conducting a controlled experiment with as little threat to validity as possible.

The present work starts with the development of a curriculum that will require students to use a directed object-orientated approach from the beginning and also to develop a system to give structure and organization for the overall programming process. In addition, special care was undertaken to introduce the teaching material in a way that limits the complexity of the Java programming language as details can easily overwhelm introductory students [16]. A group of students taught using this objects first approach is then compared with a group taught in a more conventional manner. 'Conventional' here, means an approach to learning Java programming where the basics of the programming language are first taught using procedural concepts and then the students are guided towards effective strategies for the more advanced programming skills, with direct referral to object-orientated programming coming about half way through the course. This group is used here as the control group.

Some useful development environments are now available to encourage students to approach the

* Accepted 21 June 2011.

learning of the Java programming language with a strong emphasis on the object-orientated approach [17–19] as well as web-based systems to support Java programming learning [20]. The environment chosen here for both groups was the JCreator IDE [21], which provided a simple yet professional environment with features that include project management, templates, syntax highlighting and class views. Both the experimental group and the control group used this IDE to try to eliminate bias.

In addition to the use of the above development environment, a system was incorporated to give students structure and organization during the programming process [22]. This system, derived from [23], has five stages, i.e., grouping before the computer laboratory, planning, writing the code, testing and improving. Special attention is given in this work to the planning stage where problem understanding, problem solving and the development of a strategy for writing the code are of prime importance. Here the five levels of abstraction: natural languages, diagrams, flowcharts, algorithmic languages (pseudo code) and the Java programming, all involved in the expression of algorithms [24], were given special attention.

To test for the effectiveness of the objects first approach, the results of a controlled experiment applying a pre-test–post-test control group design adopted from [25] are presented and analyzed.

The remainder of this paper is structured as follows. Section 2 contrasts procedural programming with an objects first approach, Section 3 summarizes the method and results of the controlled experiment and Section 4 reports the findings of the questionnaire. Section 5 provides a discussion of these results, including any problems with validity. The paper concludes with suggestions for improvements in the teaching and learning of the Java programming language and improvements for the design of the testing, as well as proposals for future research.

2. Software development paradigms

The two software development paradigms considered here are the procedural (sometimes called imperative) and the object-orientated.

2.1 Procedural programming

The procedural approach is 'bottom-up' and depends on the programming language, where the emphasis is placed on learning the language and not on the modeling of realistic computational problems. In addition to mastering the IDE, students usually follow a scheme of teaching such as: general program structure, declarations and variables, input/output and assignments, iterations and selections, arrays and records, functions and procedures and other features of the language. This is the approach used for the control group up to about week 6 of their 12 week course.

Understanding is also needed of the system and other diagnostics, including syntax and runtime errors, which adds another layer of difficulty in the learning process. It can be argued that the procedural approach does not teach construction of software as an engineering activity, because although engineering applications are being developed, the emphasis is not really on writing software that is suited to engineering problems, which is needed in the development of complex software.

2.2 Objects first approach

This method forces a structured approach to modular programming where the use of modules and functions establishes the principles of code re-use and functional independence. In summary, the emphasis is on modularization, encapsulation, recursion and re-use right from the beginning. This is in contrast to the procedural approach where modularity, functions and recursion are indeed part of the course, but not until the second half of the course. The objects first approach here regards the construction of software where modules and functions are the fundamental building blocks [26]. The method helps to produce properly structured and good quality modular software and is a 'top-down' approach where the important concepts of object technology are introduced right from the start.

2.3 Software development cycle (SDC)

The industry-based software development cycle [22] was adopted for the current work to improve student programming practices. Overall this cycle gives students the opportunity to program in a professional manner and will help students later in professional life as it requires the students to program following standard rules and conventions. It creates a collaborative environment for the students to work, which improves learning efficiency, communication skills and teamwork [27] and has the highly effective problem-based approach [28]. Each student has an equal opportunity to learn as the programming assignment is different for each student and a student's performance is individually evaluated.

There are five steps when implementing the software development cycle. First students are paired or grouped for a given task and two sets of problems with a similar difficulty level are given to the students. Each student in the pair takes one of the sets. The second step, planning, is where students study the problems and develop the pseudo code, flowchart or other strategies, and this is followed by

writing and self-testing the code for each of the problems individually. The fourth step is for the group partner to test the code, with students in the same group exchanging their codes and makeing cross-checks. During the checks the following points are evaluated: sufficient comments, programming conventions, programming logics and computational efficiency. On receiving the comments of their partners, the group partner improves the program according to the suggestions.

To ensure the quality of the software development cycle, each programming assignment is graded using four criteria: code correctness, the quality of the test report made for the group partner, the observance of convention with sufficient comments and on-time delivery of the code. This arrangement ensures that the students treat both their own programming assignments and their partner's seriously.

3. The controlled experiment

3.1 Description

To investigate the effectiveness of using an objects first approach to teaching and learning the Java programming language, a controlled experiment applying a pre-test–post-test control group design was conducted following [25]. The students had to undertake two tests, one before the respective course (pre-test) and one after the respective course (post-test) with the effectiveness of the teaching approaches then being evaluated by comparing within-student post-test to pre-test scores, and by comparing the scores of the students in the experimental group (A), i.e. those who were taught using the more objects first approach, to those students in the control group (B), i.e. those taught using the procedural followed by the objects method. For the objects first approach the method of delivery was by seminars and laboratories while the procedural method followed by the objects approach was delivered using lectures, tutorials and laboratories.

3.2 Hypotheses

To measure the performance of the two groups, four constructs were used, with each construct represented by one dependent variable. Each dependent variable has a hypothesis:

1. There is a positive learning effect in both groups (A: experimental group, B: control group). That is post-test scores are significantly higher than pre-test scores for each dependent variable.
2. The learning is more effective for group A than for group B, either with regard to the performance improvement between pre-test and post-test (the relative learning effect), or with regard to post-test performance (absolute learning effect). The absolute learning effect is of interest because it may indicate an upper bound of the possible correct answers depending on the method of teaching.

3.3 Method

The design started with random assignment of students to the experimental group (A) and control group (B) with the members of both groups completing a pre-test and post-test. The pre-test measured the performance of the two groups before the courses and the post-test measured the performance of the two groups after the courses. The students did not know that the post-test and pre-test questions were identical and neither were they allowed to retain the pre-test questions with the correct answers only being given to the students after the experiment.

The students were novice programmers in the second semester of an engineering course with the number of students in group A, $N_A = 23$, and in group B, $N_B = 18$. The personal characteristics of the students are summarized in Table 1.

The initial testing was conducted after a short introduction as to the purpose of the experiment and general organization issues. The pre-test was then carried out with the data for the dependent variables collected. Following the pre-test, the students were placed in either the control group or the experimental group and all students participated in both the pre-test and post-test. After completing their courses, both groups of students performed the post-test using the same questions as during the pre-test, thus providing data on the dependent variables for the second time. In addition the students were asked to answer questions about their subjective perceptions.

3.4 Teaching courses details

The syllabus used for the experimental and control groups are shown in Appendix A.

For the objects first approach a library of functions on graphics was written prior to the delivery of the Java programming module. When the students began the course their programs were written as a sequences of given functions, where students considered only the external behavior of these functions. Thus the students were helped to understand, modularization, re-use and encapsulation mechanisms, without knowing the intricacies of the computer language. It was important to get the students to execute their programs successfully early on in the course to provide a sense of confidence. The general syntax of input, output, assignment and other basic statements of the language for produ-

Table 1. Personal characteristics

Characteristic	
Average age	21.45 years
Percentage female	19%
Major	Mechanical Eng. 78% Civil Eng. 22%
Experience in computer programming	
• Never written a code	95%
• Written 1–3 codes (Language)	5% (C++)
• Written 4–6 codes (Language)	0%
Preferred learning style(s)	
• Reading with exercise	22%
• Lecture	19%
• Tutorial	26%
• Laboratory	33%
Most effective learning style(s)	
• Reading with exercise	23%
• Lecture	12%
• Tutorial	32%
• Laboratory	33%
Likes to work in groups	64%

cing basic programs were then introduced with elements of good programming style such as code readability, maintainability and functional independence also being introduced. In addition to students learning by writing programs, well structured and properly documented examples were also available to the students.

For the control group, an approach to learning Java programming where the basics of the programming language are first taught using procedural concepts was used. The students were then guided towards effective strategies for the more advanced programming skills, and, direct referral to object-orientated programming came about half way through the course

In addition a system was incorporated to give students in both the experimental and control groups structure and organization during the programming process [22]. This system, derived from Pressman [23], has five stages, i.e., grouping before the computer laboratory, planning, writing the code, testing and improving. Special attention is given in this work to the planning stage where problem understanding, problem solving and the development of a strategy for writing the code are of prime importance.

3.5 Data collection

Data for two types of variables were collected, the dependent variables (J.1, . . ., J.5) and the subjective perception variables (S.1, S.2). These variables are listed in Table 2. The dependent variables are constructs used to capture aspects of learning provided by the courses and each was measured using five questions.

The questions can be characterized as:

- J.1 ('Interest'): Questions about personal interest in learning how to program using the Java programming language.
- J.2 (Understand 'general'): Questions to elicit how much students understand the role of computer programming in engineering and generally.
- J.3 (Understand 'simple'): Questions on object-orientated Java programming that require an elementary knowledge.
- J.4 (Understanding 'difficult'): Questions on object-orientated Java programming that require a deeper knowledge.
- J.5 (Understanding 'cycle'): Questions to elicit how much students understand how to apply the software development cycle when programming.

The related questions for the subjective perceptions can be characterized as:

- S.1 ('Time pressure'): Questions on time spent completing tasks, and also about the overall length of the respective course.
- S.2 ('Course evaluation'): Questions on the students' personal judgment involving usefulness, difficulty, clarity, etc.

Selected examples of questions used are shown in Appendix B.

The results for the dependent variable J.1 were found by applying a five-point Likert-type scale [26] with each answer mapped to the value range $R = [0, 1]$.

The values for variables J.2–J.5 are average scores

Table 2. Experimental variables

Dependent variables
J.1 Interest in the Java programming language ('Interest')
J.2 General knowledge of computer programming ('Understand general')
J.3 Understanding of 'simple' object-orientated Java programming ('Understand simple')
J.4 Understanding of 'difficult' object-orientated Java programming ('Understanding difficult')
J.5 Understanding of the software development cycle ('Understand cycle')
Subjective perceptions
S.1 Available time budget versus time need ('Time pressure')
S.2 Course evaluation

derived from five questions. Missing answers were marked as incorrect.

The data for the subjective perception variables was collected after the post-test. The values for variable S.1 are normalized averages reflecting the time needed for understanding and performing the tasks associated with Weeks 2–12.

The descriptive statistics for the experiment are summarized in Table 3. The columns 'Pre-test scores' and 'Post-test scores' show the calculated values for mean ($\bar{x}$), median (m) and standard deviation (s) of the raw data collected, and the columns 'Differences' show the difference between the post-test and pre-test scores.

Table 4 shows the calculated values for mean, median and standard deviation of the raw data collected on subjective perceptions.

As can be seen from Table 4, students of the control group indicated less need for additional time to complete the different aspects of the course than those of the experimental group. Also the students in the experimental group perceived their course as easier, more engaging, clearer and more useful than the students in the control group.

Standard significance testing was used to investigate the effect of the treatments on the dependent variables J.1 to J5. The null hypotheses were:

$H_{0,1}$: There is no difference between pre-test scores and post-test scores within experimental group (A) and control group (B).

$H_{0,2a}$: There is no difference in relative learning effectiveness between experimental group (A) and control group (B).

$H_{0,2b}$: There is no difference in absolute learning effectiveness between experimental group (A) and control group (B).

For $H_{0,1}$, a one-way paired t-test was used because the data collected for this hypothesis are within-subjects, i.e. post-test scores are compared with pre-test scores of subjects within the same group [30]. For testing hypotheses $H_{0,2a}$ and $H_{0,2b}$, the appropriate test was a one-sided t-test for independent samples [30].

Focusing on the experimental group (A), Table 5 shows the results using a one-tailed t-test for dependent samples. Column one specifies the variable, two represents the Cohen effect size, d [31], column

Table 3. Scores of dependent variables

	Pre-test scores					**Post-test scores**				
	J.1	**J.2**	**J.3**	**J.4**	**J.5**	**J.1**	**J.2**	**J.3**	**J.4**	**J.5**
Group A										
$\bar{x}$	0.81	0.47	0.25	0.10	0.08	0.84	0.88	0.78	0.41	0.53
m	0.85	0.38	0.28	0.09	0.09	0.83	0.96	0.74	0.37	0.51
s	0.11	0.28	0.27	0.21	0.24	0.13	0.13	0.12	0.15	0.13
Group B										
$\bar{x}$	0.76	0.46	0.29	0.14	0.11	0.77	0.51	0.66	0.37	0.32
m	0.75	0.44	0.28	0.16	0.08	0.79	0.49	0.68	0.37	0.30
s	0.14	0.19	0.25	0.23	0.22	0.23	0.11	0.13	0.18	0.21
	Differences									
	J.1		**J.2**		**J.3**		**J.4**		**J.5**	
Group A										
$\bar{x}$	0.03		0.41		0.53		0.31		0.45	
m	*–0.02*		0.58		0.46		0.28		0.42	
s	0.12		0.22		0.21		0.18		0.19	
Group B										
$\bar{x}$	0.03		0.05		0.37		0.23		0.21	
m	*–0.02*		0.05		0.40		0.21		0.22	
s	0.57		0.16		0.20		0.21		0.22	

Table 4. Scores of subjective perceptions

	S.1	S.2
Group A		
$\bar{x}$	0.41	0.53
m	0.43	0.54
s	0.32	0.17
Group B		
$\bar{x}$	0.39	0.42
m	0.40	0.39
s	0.27	0.15

Table 5. Results for 'post-test' versus 'pre-test' for group A

Variable	*d*	df	t-value	Crit. $t_{0.90}$	p-value
J.1	0.249	22	1.194	1.321	0.249
J.2	1.878	22	6.572	1.321	0.000
J.3	2.536	22	7.637	1.321	0.000
J.4	1.698	22	6.249	1.321	0.000
J.5	2.331	22	7.322	1.321	0.000

Table 6. Results for 'post-test' versus 'pre-test' for group B

Variable	*d*	df	t-value	Crit. $t_{0.90}$	p-value
J.1	0.053	17	0.977	1.333	0.171
J.2	0.322	17	2.407	1.333	0.013
J.3	1.857	17	5.782	1.333	0.000
J.4	1.114	17	4.478	1.333	0.000
J.5	0.976	17	4.191	1.333	0.000

three the degrees of freedom, column four the t-value of the study, column five the critical value for the significance value a = 0.10 and column six lists the associated p-value. Using the suggestions of [25], testing for the normality assumption, analysis to detect outliers and the non-parametric tests of Wilcoxon and the Mann-Whitney U test were carried out for the hypothesis $H_{0,1}$ and for the hypotheses $H_{0,2a}$ and $H_{0,2b}$ respectively. It was found that no normal distribution of the variables could be assumed and that all the data lay within the ±2 standard deviations around the samples' means. The non-parametric tests did not show any difference from the results of the t-tests.

It can be seen from Table 5 that the experimental group A achieved a statistically and practically significant result for the dependent variables J.2–J.5, whereas J.1 did not achieve a significant result.

Table 6 shows the results of testing Hypothesis $H_{0,1}$ for the control group (B) using a one-tailed t-test for dependent samples. The structure of the table is the same as that of Table 5.

Again for this group the dependent variables J.2–J.5 achieved statistically and practically significant results, whereas the dependent variable J.1 did not.

Hypothesis $H_{0,2a}$, which states that the difference between the post-test and pre-test scores of group A is not significantly larger than those of group B is not examined. Table 7 shows, separately for each dependent variable, the results of testing hypothesis $H_{0,2a}$ using a one-tailed t-test for independent samples.

Table 7. Results for 'performance improvement' (Group A versus B)

Variable	*d*	df	t-value	Crit. $t_{0.90}$	p-value
J.1	0.000	39	0.000	1.304	0.500
J.2	1.872	39	8.653	1.304	0.000
J.3	0.780	39	5.586	1.304	0.000
J.4	0.409	39	4.045	1.304	0.000
J.5	1.168	39	6.835	1.304	0.000

Table 8. Results for 'post-test improvement' (Group A versus B)

Variable	*d*	df	t-value	Crit. $t_{0.90}$	p-value
J.1	0.375	39	3.872	1.304	0.000
J.2	3.073	39	19.435	1.304	0.000
J.3	0.959	39	6.194	1.304	0.000
J.4	0.241	39	3.105	1.304	0.002
J.5	1.202	39	6.934	1.304	0.000

It can be seen from Table 7 that the hypothesis $H_{0,2a}$ can be rejected for the dependent variables J.2–J.5, and for these dependent variables their results support the direction of the expected relative learning effect. For the dependent variable J.1, the result was not statistically and practically significant and did not even support the direction of the hypothesis.

Table 8 shows, separately for each dependent variable, the results of testing $H_{0,2b}$ using a one-tailed t-test for independent samples. The hypothesis $H_{0,2b}$ can be rejected for the variables J.1–J.5, showing the expected relative learning effect to be strongly supported both statistically and in practice.

4. Questionnaire results

A questionnaire regarding the difficulty of learning eight programming design features, as implemented in Java was given to both the experimental and control groups, who had completed their courses. Four of the subjects gave no responses. On a scale of 1(easy) to 5 (very difficult) the subjects were asked to rate the difficulty in learning program/design features of Java in the following: Syntax, Objects, Methods, GUIs, Arrays, Exceptions, Inheritance and Polymorphism. The ratings given are subjective in that what is being reported is what the subjects perceived as 'easy' or 'difficult' to learn. Comparisons were made between the experimental and control group responses.

Figure 1 summarizes the results of t-tests. Levene's 'test for equality of variances' [32] allows the user to determine whether the variances from the two groups are equal or unequal, and gives the p-values for each case.

From Fig. 1 and Table 9 it can be seen that, except for 'Syntax', 'Objects' and 'Arrays', the tested

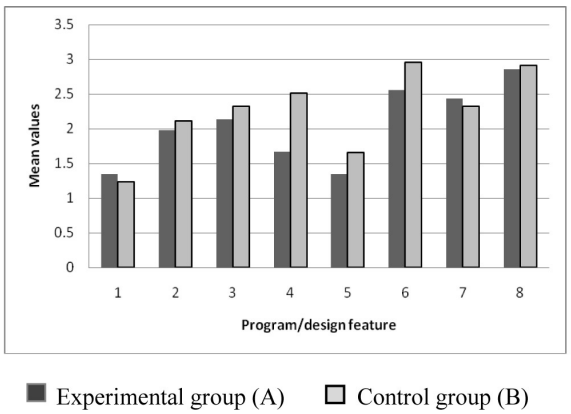

Fig. 1 Comparison of mean responses of the experimental group (A) and control group (B): (1) Syntax; (2) Objects; (3) Methods; (4) GUIs; (5) Arrays; (6) Exceptions; (7) Inheritance; (8) Polymorphism.

Table 9. p-values for experimental group (A) and control group (B)

Program/Design feature	p-value
Syntax	*0.0879*
Objects	0.1052
Methods	0.0054
GUIs	0.0043
Arrays	0.2312
Exceptions	0.0007
Inheritance	*0.0012*
Polymorphism	0.0765

features were significantly ($p < 0.10$) perceived as more difficult to learn by the control group (B) than for the experimental group (A). The control group (B) perceived the learning of 'Syntax' and 'Inheritance' as being easier.

5. Discussion of results

5.1 Controlled experiment

When considering the positive learning effect within the experimental group (A), a statistically significant positive change of scores was found from the pre-test to post-test for the dependent variables J.2 to J.5. On the other hand, no positive effect could be found for variable J.1, showing that the course had not aroused interest in the Java programming language. A similar result was found for the control group (B), where again all dependent variable scores improved except for J.1 'Interest'. The reasons for this initial lack of interest and lack of growth of interest could be many. These could range from 'students do not really see computer programming as an important part of an engineering course' to 'the fact that programming is a highly demanding and detailed procedure'.

With the exception of the variable J.1, testing the performance of relative and absolute learning effectiveness between groups (hypotheses $H_{0,2a}$ and $H_{0,2b}$) showed that the more object-orientated course yielded significantly better scores for the relative effectiveness and significantly better results for the absolute learning effectiveness for all dependent variables.

It was noted that the extra time needed to introduce and implement the software development cycle, which gave additional time pressure on the subjects in the experimental group (A) did not have any negative effect on the scores of variables J.3 and J.4. It was also noted that implementation of the software development cycle also produced more cooperation and teamwork among students.

5.2 Questionnaire

From Fig. 1 and Table 9 it can be seen that, with the exception of 'Syntax', the subjects within the experimental group (A) perceived the acquisition of knowledge and skills of the Java programming language as being easier than those of control group (B). Generally 'Syntax' and 'Arrays' were seen as relatively simple, whereas 'Exceptions' and 'Polymorphism' were seen as difficult. It is noteworthy that the experimental group only perceived 'Objects' and 'Methods' as only marginally easier when compared with the control group, and this could imply that the more object-orientated approach is not too difficult for novice students.

5.3 Validity

The validity of the present work is now discussed. It is first recognized that interest in a topic and evaluation of a teaching session are difficult to measure, and to alleviate this problem the instruments for measuring variables J.1 and S.2 were derived from measurement instruments that have already been successfully applied in similar kinds of studies [25, 33].

To alleviate selection threats when dividing the students into two groups, a randomization procedure was used. This, together with the student characteristics of similar age, level of programming experience and general level of education when starting each programming course gave reasonable assurance of minimum bias. Also, there was no change in teaching staff throughout the courses, so reducing any 'selection history effect', and, as none of the subjects left their respective group, there was no 'dropout interaction effect'. Students in both groups were asked not to discuss their course with members of the other group to try to reduce the 'diffusion or contamination' effect. Also, it was mentioned to all the students that each course was a legitimate method of acquiring their programming skills and one course was not necessarily better than the other. This was an effort to minimize any 'rivalry or resentment' threats.

After selection any differences in the ability of the groups was captured by collecting pre-test scores. The subjects were all novices, so it can be expected that they are representative of a general undergraduate population.

6. Conclusions

The empirical studies presented in this paper investigated the use of a more object-orientated approach to the teaching and learning of the Java programming language. It was found that this new approach helped student in the areas of general knowledge of computer programming, understanding of 'simple' object-orientated Java programming, of 'difficult' object-orientated Java programming, and of the software development cycle.

The level of interest in the Java programming language was not raised by either the more object-orientated course nor the conventional course and this is a subject for future research and innovation, as is the question 'would the results be different if a different computer programming language was investigated?'.

References

1. M. Kölling, The problem of teaching object-orientated programming, Part 1: Languages, *Journal of Object-Orientated Programming,* **11**(8), 1999, pp. 8–15.
2. J. Carter and A. Jenkins, The problems of teaching programming: Do they change with time? *The Higher Education Academy, Information and Computer Sciences, 11th Annual Conference*, August 24–26, 2010, Durham, UK.
3. I. Grovender, and D. J. Grayson, Pre-service and in-service teachers' experiences of learning to program in an object-oriented language, *Computers & Education,* **51**, 2008, pp. 874–885.
4. Q. H. Mahmoud, W. Dobosiewicz and D. Swayne, Redesigning introductory computer programming with HTML, JavaScript and Java, *Proceedings of the 35th ACM Technical Symposium on Computer Science Education (SIGCSE)*, March 3–7, 2004, Norfolk, Virginia, USA, pp. 120–124.
5. M. McCracken, V. Almstrum, D. Diaz, M. Guzdial, D. Hagan, Y. B. D. Kolikant *et al*., A multi-national, multi-institutional study of assessment of programming skills of first year CS students, report by the ITiCSE 2001 Working group on Assessment of Programming Skills of First-year CS students, 2001.
6. I. Milne and G. Rowe, Difficulties in learning and teaching programming—views of students and tutors, *Education and Information Technologies,* 7(1), 2002, pp. 55–66.
7. A. Benander, B. Benander and J. Sang, Factors related to the difficulty of learning to program in Java—an empirical study of non-novice programmers, *Information and Software Technology*, **46**, 2004, pp. 99–107.
8. D. Adair and M. Jaeger, Developing programming skills to enhance engineering dynamics modeling, *Proceeding of the IETEC'11 Conference*, 16–19January, 2011, Kuala Lumpur, Malaysia.
9. D. Clark, C. MacNish and G.F. Royle, Java as a teaching language—opportunities, pitfalls and solutions, *Proceedings of the Third Australian Conference on Computer Science Education, ACM*, July, 1998, Brisbane, Australia.
10. M. Kölling and J. Rosenberg, Guidelines for teaching object orientation with Java, *Proceedings of the 6th Conference on Information Technology in Computer Science Education*, 2001, Canterbury, UK.
11. A. Robins, J. Rountree and N. Rountree, Learning and teaching programming: A review and discussion. *Computer Science Education,* **13**(2), 2003, pp. 137–172.
12. E. Lahtinen, K. Ala-Mutka *et al.,* A study of the difficulties of novice programmers, *10th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education, ITiCSE'05*, 2005.
13. S. Wiedenbeck and V. Ramalingam, Novice comprehension of small programs written in the procedural and object-oriented styles *Int. J. Human–Computer Studies*, **51**, 1999, pp. 71–87.
14. C. Corritore and S. Wiedenbeck, An exploratory study of program comprehension strategies of procedural and object-oriented programmers *Int. J. Human–Computer Studies,* **54**, pp. 1–23.
15. G. White and M. Sivitanides, Cognitive difference between procedural programming and object oriented programming *Information Technology and Management*, **6**, 2005, pp. 333–350.
16. Joint Task Force on Curricula, Computing Curricula 2001 Computer Science, *Journal of Educational Resources in Computing*, **1**(3): entire issue, 2001.
17. R. Lister, Teaching Java first: experiments with a pigs—early pedagogy, *Australasian Computer Education, 6th Conference,* 18–22January, 2004, Dunedin, NZ.
18. K. Ala-Mutka, Problems in learning and teaching programming—a literature study for developing visualizations in the Codewitz-Minerva project, 2005, Retrieved September 30, 2010, from www. cs.tut.fi/~codewitz/literature_study.pdf.
19. T. Wang, X. Su, P. Ma, Y. Wang and K. Wang, Ability-training-orientated automated assessment in introductory programming course, *Computers & Education*, 2011 (article in press).
20. A. Mendes, V. Ivanov and M. Marcelino, A web-based system to support Java programming learning *International Conference on Computer Systems and Technologies, CompSys Tech' 2005*, 2005.
21. JCreator, Xinox Software, Delft, Netherlands, www.jcreator.com, 2010.
22. W. P. Sun, A new paradigm to improve computer education for engineering students: Applying industry-based software development cycle into programming practices, *ASEE Annual Conference*, June 24–27, 2007, Hawaii.
23. R. S. Pressman, Software engineering—a practitioners' approach, 6th edition, McGraw-Hill, New York, 2004.
24. J. P. Tremblay and R. B. Bunt, *An Introduction to Computer Science: An Algorithmic Approach*, McGraw-Hill, New York, 1989, pp. 18–23,
25. D. Pfahl, O. Laitenberger, G. Ruhe, J. Dorsch and T. Krivobokova, Evaluating the learning effectiveness of using simulations in software project management education: results from a twice replicated experiment, *Information & Software Technology,* **46**, 2004, pp. 127–147.
26. R. Bornat, *Programming from First Principles*, Prentice Hall, 1986.
27. R. Pimmel, Cooperative learning instructional activities in a capstone design course, *Journal of Engineering Education,* **90**(3), 2001, pp. 413–421.
28. W. P. Sun and J. Anderson, Teaching plant design/material handling by using project-based approach, *Proceedings of the 2006 ASEE Annual Conference & Exposition*, 2006.
29. R. Likert, A technique for the measurement of attitude, *Archives of Psychology,* **22**(140), 1932.
30. D. Sheskin, *Handbook of Parametric and Nonparametric Statistical Procedures*, CRC Press, Boca Raton, 1997.
31. J. Cohen, *Statistical Power Analysis for the Behavioral Sciences*, 2nd edn, Lawrence Erlbaum Associates, Hillsdale, NJ, 1988.
32. H. Levene, Robust tests for equality of variances. In I. Olkin (ed.), *Contributions to Probability and Statistics, Essays in Honor of* Harold Hotelling, Stanford University Press, 1960, pp. 278–292.
33. J. A. M. Vennix, Mental models and computer models: design and evaluation of a computer-based learning environment for policy making, Doctoral Dissertation, Katholieke Universiteit Nijmegen, 1990.

Appendix A: Teaching syllabuses

A.1 Java programming course syllabus

- Content
- Unit introduction, Programming terms & tools, Computing terms & tools
- Solving problems with computers. Data storage primitive types, Data storage manipulation of primitive data
- Using class libraries, Object methods, Class methods
- Flow of control branches, Planning & implementing branches, Multi-way branching
- Flow of control loops, Implementing loop algorithms, Nesting flow
- Extending existing classes, Parameters & return values, Extending classes, Graphical User Interface
- New classes simulating real world objects, Implementing and using new classes, Testing & Documenting programs
- New classes to organize tasks, Method decomposition, GUI interface adding components
- Structured data, Arrays, Declaring & filling arrays, Using arrays inheritance polymorphism
- Standard array algorithms, Sorting algorithms, Searching algorithms
- GUI interactivity event driven programs, GUI implementation events, Input from the GUI
- Run time errors exceptions, Recursion concepts, Recursion implementation

In addition, two major assignments, done on an individual basis, were completed by the students.

Appendix B. Example questions

Dependent variable J.1

I consider it very important for engineering students to know as much as possible about computer programming. (1 = fully agree / 5 = fully disagree) Circle number below.
Agree 1 2 3 4 5 Disagree

Dependent variable J.2

Computer software is important to the following areas of engineering:
Control engineering True/False
Thermodynamics True/False
Materials True/False

Dependent variable J.3

What is the output of the following code?

```
public class Example {
  public static void main(String[] args) {
  int num = 3;
  if (num == 3) {
  System.out.println('Hello world!');
              }
       }
  }
```

Dependent variable J.4

What can go wrong if you replace && with & in the following code:

```
  String a=null;
  if (a!=null && a.length()>10)
  {...}
```

Dependent variable J.5

What is the developer's goal?

Subjective perception S.1

I did not have enough time to:
- complete the tutorials
- complete the computer laboratory sessions
- write up the assignments
- complete the post-test

Subjective perception S.2

I consider the explanations/ provided for the computer laboratory sessions

1 2 3 4 5	
Useful	Useless
Engaging	Boring
Easy	Difficult
Clear	Confusing

Desmond Adair holds a Ph.D. in Mechanical Engineering from Imperial College, London. He spent a number of years working as a Senior Research Engineer with NASA in California and NPL in Teddington, England. Dr. Adair has worked for British Aerospace and the UAE Defence Forces in senior education positions, and is a Research Associate with the University of Tasmania, Australia.

Martin Jaeger holds a Ph.D. in Civil Engineering (Construction Economy and Management) from the University of Wuppertal, Germany. He spent the last ten years working as site manager, consultant, and lecturer in Germany and the Middle East and is a Research Associate with the University of Tasmania, Australia.

Jonathan Stegen is a graduate of the University of Canterbury, New Zealand and has worked in many countries including Japan, England and Saudi Arabia. He has also worked as a consultant for Rusal (Russian Aluminium) in Moscow and is at present a Senior Instructor at the Australian College of Kuwait.