From Procedural to Object-Oriented Programming (OOP) - An exploratory study of teachers' performance

Irene Govender

School of IS & T, University of KwaZulu-Natal

Abstract

This exploratory study of introductory pre- and in-service teachers' performance in object-oriented programming (OOP) assessments reveals important issues with regard to learning and teaching OOP, using Java. The study is set against the backdrop of the country's transition of its national IT curriculum from a procedural to an object-oriented programming language. The effect of prior programming experience and performances in different types of questions are examined. A combination of quantitative and qualitative methods is used to analyse the data. The effect of prior programming experience of a procedural kind and the type of assessments given is shown to have a marked influence on the performance in programming assessments and teaching of OOP. Many introductory OOP courses are in effect taught procedurally as courses in the small. Therefore educating teachers how to teach programming is an important educational challenge. Some implications for teaching are therefore suggested.

CATEGORIES

K.7.0, 1

KEYWORDS

Procedural programming; object-oriented programming; assessment; pedagogical content knowledge

1. INTRODUCTION

The national curriculum statement (NCS) for Information Technology in South Africa [8] proposed a new programming language that is object-oriented (OO). The change in programming language has implications for teacher-trainees and for many in-service teachers, who learnt and were trained to teach a procedural language, such as Pascal. Several studies (see, for example [13]) show that programming is a challenge for introductory students. By studying the programming assessments of pre- and in-service teachers during the transition from procedural to object-oriented programming (OOP) revealed the influences prior knowledge has on learning and teaching OOP. Since learning and teaching could be regarded as two sides of the same coin, knowledge of students' performance in programming assessments and their thinking processes of learning to program can inform teachers' instruction. The purpose of this study is therefore to explore the ways in which novice teachers (includes both those teachers that are new to programming and those that are new to OOP, but who may have had experience in procedural programming) of programming learn and perform in assessment questions based on object oriented programming.

2. LITERATURE SURVEY

There is general agreement in the literature that learning to program is not an easy task [13, 14]. Having to program in a new style if one has prior programming experience creates tensions between learning to program and learning to teach programming. Teachers' perceptions and behaviour are formed by their own experiences, both their past experiences and current views. Bergin and Winder [2], among others, believe that OOP is a paradigm, different from procedural programming, which requires a change in mental model (a paradigm shift) in the practitioners. After being used to a procedural style of programming, learning to program in an object-oriented style seems to be very difficult. For example, Stroustrup [21] indicates that it takes an average programmer 6 to 18 months to switch the mind-set from procedural to object-orientation. What it may be pertinent to ask now is, will programming in the new paradigm be just as, or less, difficult as it is in the old paradigm, or does the shift in paradigm pose additional difficulties?

The myth that "object-orientation and procedural concepts are mutually exclusive" is refuted by Lewis [15]. He argues that an object-oriented approach does not throw out the concepts that are admired in a procedural approach; rather it augments and strengthens them. A number of recent studies [3, 5] explore issues relating to the OOP paradigm. The argument presented for embracing the OO approach is twofold. In the first instance, it is argued that objects are natural features of problem domains and can be represented as entities in the programming domain. Secondly, the mapping between the domains is simple and should, therefore, support and facilitate OOP design. However, the literature reviewed shows that identifying objects is not an easy process for novices, and the mapping between domains is not straight forward. While the literature on expert programmers is supportive of the naturalness and ease of OO design, it also shows that expert OO programmers use both OO and procedural views of the programming domain, and switch between them as and when necessary [9]. However, this study is particularly concerned with novice OO programmers.[1] In their study, Bergin and Reilly [3] found that among the factors that influence programming success, self-perception of course outcomes were the most strongly correlated to performance.

In a separate study, Wiedenbeck and Ramalingam [23] found that first year tertiary students' self-efficacy of programming is influenced by their previous secondary school programming experience, which in turn influence their performance. In this study, however, previous programming experience of the participants is explored in relation to learning a *different* programming paradigm, OOP. Therefore the experience reported in this study would help further understand the learning of OOP and the teaching of OOP by novices, which in turn adds to the body of literature.

2.1 Teaching of OOP

Several studies have proposed approaches to teach OOP [16, 11]. However, few have proved one method to be more successful than another. Bennedsen and Caspersen [1] believe that

> …the learning of programming should be embedded in a context where the primary focus is learning systematic techniques to develop a program from a conceptual model of the problem domain and to apply these techniques [1]

Kölling [14] agrees that teaching OOP seems to be more difficult than teaching procedural programming. Ritzhaupt and Zucker [17] further supports this notion by advocating teaching OOP in a second programming course in which the objects-first approach is used. While the approach to follow when teaching the programming language is of concern, it must be noted that it is a national imperative to teach an OOP language.

An important trend in the literature is the distinction between studies that explore program comprehension (in which students are given the code of the program, and for which they have to explain or demonstrate their understanding of the code), and those studies that focus on program generation (in which students have to create a part of, or a whole program to perform a task or solve a problem).

[1] Note that novice OO programmers may have had experience in procedural programming and are, therefore, not necessarily the same as completely novice programmers.

Reading and understanding code is an important aspect of learning to program. Deimel (cited in [22]) believes that this skill should be explicitly taught. Studies to identify misconceptions in object-oriented courses have received a great deal of attention recently. In their study, Sanders and Thomas [19] have discussed important misconceptions in OOP. I believe that the exercise of reading and understanding code is important in unveiling some of the misconceptions that would in turn help alleviate the problem of understanding and implementing OOP concepts

An examination of the literature indicates that there are more studies of comprehension of programs than there are of generation of programs [18]. Robins, Rountree and Rountree [18] suggest that this might be because comprehension studies are generally more narrowly focused and controlled, and it is, therefore easier to understand and explain the students' behaviour. However, it is clear that program comprehension and program generation are related, because during generation the development, debugging (and, in the long term, maintenance) of code involves reviewing and understanding it. One expects these abilities to be highly correlated; however, there are more issues to consider before a direct correspondence can be made. In terms of drawing a direct comparison of comprehension and generation (reading and writing) type questions, Simon, et al. [20] acknowledges the difficulty of assessing the comparability of reading and writing questions.

> It is not clear to us that this distinction between line-by-line understanding and big-picture understanding has a parallel in code-writing questions [20].

This issue is still of concern in current studies. In teaching students to program, it would therefore be necessary to ask, which should be emphasized more. In this study, the issue is explored again to see how this particular experience with different programming backgrounds adds to or detracts from the clarity referred to by Simon, et al. [20]. Teaching and learning to program in OO requires one to consider the efficacy of the different types of assessment questions. This study is crucial in that the transition from a procedural language to an object-oriented language, as experienced by pre- and in-service teachers can play a major part in our understanding of the difficulties and successes in learning and teaching the new language.

In order to gain insights into students' performance in object-oriented programming, I attempted to answer the research questions stated in the next section.

3. RESEARCH METHODOLOGY

3.1 Research questions

1. Is prior programming experience a predictor of success for teaching and learning the new object-oriented language?
2. How do (pre- and in-service) teachers differ in their ability to answer questions on comprehension and generation of code?

3.2 Research participants

Convenience sampling of students was used in the study. 85 in-service teachers (most of whom were teaching Pascal at secondary schools) enrolled for a year-long course in Java at a distance learning institution, and 14 pre-service teachers that are studying towards a teaching qualification at a local university and who chose to major in computer science education studied programming over two semesters (14 weeks per semester). These students attended face-to-face lectures.

3.3 Format of course for both groups

The course consisted of general programming and basic object-oriented concepts such as methods, objects, classes, instantiation, constructors and program flow. These topics were common to both courses, however, the two groups were taught by different instructors.
Both groups wrote formal examinations at the end of the course. These assessments, together with informal discussions with instructors and journal writing were used in the analysis. The duration of the examinations was 3 hours for a total of 100 marks. Students were expected to do the following:

- Understand Java code which was given;
- Write Java code
- Answer simple theoretical questions on Java
- Answer questions relating to pedagogical content knowledge topics covered in the study guide.

3.4 Data Collection

The primary data source for this study comprised the examination paper for both groups of students and journals that students were required to keep as part of both courses.

Participants' reflections of their experiences in teaching programming and learning a new programming language were reflected in their journals. The journals served as records of growth and were the main source of information about in-service teachers' learning processes. The journals were part of the compulsory course assessment for the in-service teachers. Other than the specific activities which had to be done in the journal, students were asked to reflect on their experiences as they worked through the course material. Examples of entries to be written in the journals were thoughts and ideas, questions and problems, feelings, notes, ideas on how to teach, and specific comments on the course material.

In assessing the journal, marks were awarded based on how regularly they wrote in their journals, and how much careful thought, honesty and effort went into writing in the journal. There were no right or wrong answers. Such journals are useful tools to understand the mental processes that students engage in as they read, write and problem solve [7].

With consent of the teachers, a background questionnaire was used to draw responses with respect to previous knowledge of programming, together with the number of years either teaching or using the language (with regard to in-service teachers). A similar questionnaire was administered to the pre-service teachers as well. The in-service teachers were also asked to rate their experience of programming with a specific language, as: *limited knowledge*, *know the basics* and *know the language well.* Using SPSS, a statistical analysis program, the data of students' raw scores obtained in the examination together with their programming knowledge, were captured. Their programming knowledge was coded according to the level of experience and number of years teaching/using the language and thereby rated on a scale from 0 to 5 according to Table 1.

I anticipated a deeper understanding of learning to program as experienced by the candidates by including both in-service and pre-service teachers in this study. In general in-service teachers, who were experienced in procedural programming (in this case, Pascal), are insightful and contribute valuable material for stimulating reflections on teaching [4].

Table 1: Rating scale of programming experience

Know the language well – teaching/ using more than 5 years	Know the language well – teaching/ using 1-4 years	Know the basics – teaching/ using more than 8 years	Know the basics- teaching/ using 1-4 years	Know the basics- teaching/ using 0 years	Limited knowledge of Pascal	Limited knowledge of any other language	none
5	4	4	3	2	2	1	0

3.5 Questions chosen for discussion

Because of the exploratory nature of this study, the questions chosen for analysis and discussion were limited to the examination paper. For the purposes of this study 2 questions were selected from each examination paper with the view of discussing the comprehension and generation of code.

3.6 Analysis

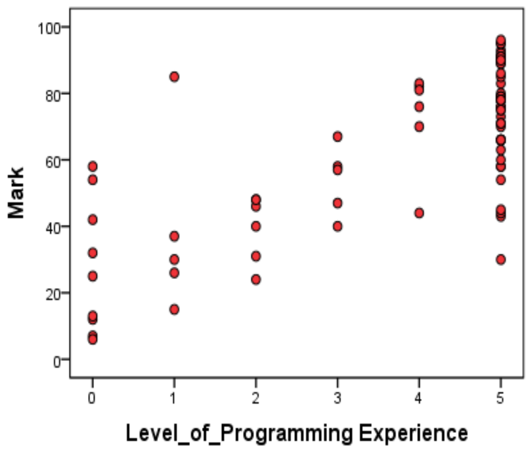
Both quantitative and qualitative analysis was used in this study. The informal discussions with instructors and the journal writings were analysed for common themes and issues regarding the aspects tested within the examination. In the analysis and discussion sections quotations from participants' journals will be provided in support of claims made.

4. ANALYSIS AND DISCUSSION

The assessments for the two sets of participants will be dealt with separately.

4.1 Performance in assessments of in-service teachers

The raw scores obtained in the examination of the (85) in-service teachers, and the corresponding level of programming background, are illustrated in the graph below.



On close examination of the graph, it is evident that there are general clusters of circles/points. The levels of programming experience rated as 3, 4 and 5 are associated with percentage marks above 50% and level of programming experience rated as 0, 1 and 2, are associated with percentage marks of below 50%. The mean percentage marks for each level of programming was calculated. It was found that there is an association between the level of programming experience or exposure, and the average percentage mark.

The mean percentage marks of 27, 39 and 40 correspond to prior level of experiences rated as 0, 1 and 2 respectively, while the mean percentage marks of 55, 72 and 75 are associated with prior level of experiences rated as 3, 4 and 5 respectively. This result seems to indicate that previous programming experience or knowledge is a predictor of success in learning a specific new programming language, Java, which concurs with the literature. It is important to note at this stage that the participants' previous programming experience is to a large extent based on procedural programming using Pascal. On the surface, it would appear that shifting from a procedural paradigm to an objected-oriented paradigm is achieved with relative ease. Those in-service teachers who scored well had an advantage of past programming concepts and principles as a kick start to the course. Learning the details of programming, such as variables, loops, if…then…else, arithmetic and Boolean operators were familiar to them. In this regard, all they needed to get used to was the new syntax. This result seems to fly in the face of the general literature (see for example, [21]) that indicates that students with a procedural background will take a longer period of time to make a shift to the object-oriented paradigm. However, a detailed examination of the questions and data reveals more than is immediately evident. The problem task indicated breaks up the task into neat little units that need to be implemented or if asked to solve a complete problem (as in the case of question 2) then only a one-class program is required. This certainly favours the students who have had strong procedural programming background knowledge.

The following quotations, from the journal entries of in-service teachers, indicate the dependence on hints and small chunks of code, which they have been using in their teaching. Similar characteristics are used in the examining process of the in-service teachers.

> All pupils prefer, in a test situation, if you break the problem into parts and/or give hints. They are more stressed when doing exams/tests than class-work and giving them the "starting board" to the solution reduces the stress for them
>
> I do break down the bigger problems into parts – more to ensure clarity of the question and to clearly set out the major objectives of the problem.

What is pertinent to point out at this stage is that, although the quotations above are from in-service teachers studying this course, many of them are also teachers of the subject matter, albeit of a procedural language. There is a strong possibility that they will carry this mode of teaching and assessment into their classrooms. The students are still programming in the small. It is not clear whether or not they have mastered the true OO design principles and characteristics of OOP. The graph does suggest that most participants who have a level of experience of 3 and above seemed to have scored above 50%. It is reasonable to assume, therefore, that prior programming experience is helpful for programming in the "small".

The in-service teachers who did not score well (below 60), and who are associated with low levels of prior programming experience, had to make an enormous effort in order to pass. In addition to learning the basic programming structures (looping, if ….then…else, variables, etc), they had to learn the concepts of objects, classes, inheritance, constructors etc. In a short space of time, the novice learners needed to go through a steep learning curve. Other contributory factors that may have hindered their performance are the problems associated with distance learning such as, being employed full-time as teachers and not least of which is this novel way of thinking in programming. Possibly, given a longer period to follow these students may show different results. To answer the question posed above (*Is prior programming experience a predictor of success for the new object-*

oriented language?), yes, prior *procedural* programming experience is a predictor of success for the object-oriented *language*; however, no; prior *procedural* programming experience is not necessarily a predictor of success for object-oriented *programming*. This would become clearer when the performance in different questions are examined in the next section. Note that the emphasis is the difference between success in learning the OO *language*, and success in learning the style of OO *programming*. This result suggests that the introductory course concentrates on the procedural aspects of programming in Java and the OO aspects are to a large extent neglected. Hence students' experience of the introductory course seems to have a procedural bias, rather than object-oriented. Their "history" (previous learning) of programming experience certainly has an influence on their learning.

4.2 Secondary analysis of the performance in specific questions

This section considers the qualitative difference in performance in different questions to try to answer the following question:

How do students differ in their ability to answer comprehension (tracing) type questions and generation of code type of questions?

For this question, both groups of teachers (pre-service and in-service) were considered separately.

4.2.1 Analysis of Selected Solutions of In-service Teachers

On inspection of the in-service teachers' examination, key observations were made. Questions 2 and 5 (see Appendix for the questions referred to) were specifically analysed because they relate to questions on code generation and comprehension respectively. The mark obtained for questions 2 and 5 have been extracted from the data.

Table 2 summarizes the means and standard deviations of the surveyed students on the two questions. All marks are unscaled (raw) and are expressed as percentages.

Table 2: Descriptive statistics for in-service students' performances in different questions

	Variable Mean (%)	Standard Deviation (%)	Cases(N)
Q2	86	24	85
Q5	53	33	85

Positive correlations were found between students' marks obtained in question 2 and the final mark. The Pearson's correlation matrix is shown below in Table 3. All correlations are statistically significant (p<0.01).

Table 3: Pearson's correlation coefficients for assessment in programming questions

	Q2	Q5
Q2	1	
Q5	0.584003	1

There was a positive correlation of 0.584003 (p<0.01) between students' scores in question 2 (Q2) and question 5 (Q5). While it is positive, it is not as strong as expected. One would assume that good performance in questions on code generation would surely imply good performance in questions on code comprehension [20]. What does this tell us? Firstly, comprehension of the answers and hence explanations required to substantiate them, is not strong enough. This concurs with [20]. Question 5 required students to use "variable box diagrams and arrows" (memory diagrams, which represents the state of objects in memory at a particular point in the execution of a program.) to explain the problem in the given code. Hence, a thorough understanding of object-oriented concepts such as local variables, static methods and variables, and memory allocation of objects is necessary. The lack of adequate ability to use these memory diagrams is suggestive of a poor understanding of the concepts. In effect, it was a debugging exercise. While literature suggests that it might be easier to comprehend (explain, edit or modify) existing code, in this particular instance, contrary to the implication in the literature, students performed better at generation of code (Q2) than in the comprehension of code (Q5). The mean scores of Q2 and Q5 in Table 3 reflect this scenario. However, on examining the question in detail, it suggests that question (Q2) was a simple, "common" calculation which required a one-class program. For convenience I will reproduce a segment of code from question 5 here:

Question 5

```
public class UseDate
        {
          static Console c;           // The output console
          public static void main (String [] args)
          {
            c = new Console ();
            c.print ("Enter year, month, day (separated by
                                                    spaces) :");
            int year = c.readInt();
            int month = c.readInt();
            int day = c.readInt();
            Date userDate = new Date (year, month, day);
            if (userDate.isLeap())
            c.println("Is leap year");
           else
            c.println("Is not leap year");
          } // main method
        }    // The "UseDate" class.
```

Depending on the year entered by the user, the program should display "Is leap year" or "Is not a leap year". The problem is that no matter what the user types in, the program always displays "Is leap year".

5.1 Explain why the program is not working as it should
and what can be done to fix the problem

The understanding of program flow, the necessary assignments to variables and the instantiation of objects using constructors with parameters seem to be problematic for many students. This is suggested by the mean score for question 5.

The memory diagrams are certainly useful for understanding object references in a code fragment. Even the in-service teachers, who have had sufficient experience with learning and teaching procedural programming, have found the memory diagrams very useful. This is indicated by the following quotations from the in-service teachers' journal entries:

> In my experience of teaching variables I found that learners have extreme difficulty in understanding the concepts of a variable, that is, how variables are kept in memory. I think using variable box diagrams to teach the concept of a variable is very effective. ….many learners grappled with the understanding of how variables are stored in memory especially when a variable takes on a new value. The box approach has sorted out this difficulty that learners were experiencing.

An example of a memory diagram is shown in Figure 2.

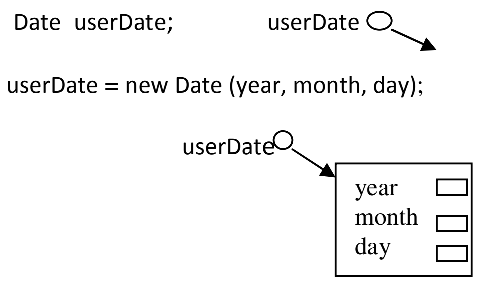


Figure 2 Representation in memory

Another quote from a journal entry follows:

> It is also similar to tracing through programs which helps learners see exactly what is happening at each statement and understand how the program works together with seeing the exact output.

While the above quotations are with respect to procedural programming (the in-service teachers were teaching Pascal and learning Java), they certainly are true for object-oriented programming as well, in which more abstract references are required. Memory diagrams, with respect to object references, still continue to be problematic, which concurs with the reviewed literature [10].

4.2.2 Analysis of Selected Solutions of Pre-service Teachers

In order to better understand some of the outcomes and results in the previous section, a similar analysis of the solutions of the pre-service teachers was performed. Two questions (4 and 6) from the examination were chosen for similar reasons stated in the previous section. These questions were similar in nature to questions 2 and 5 respectively. For both examinations, the code generation questions were straight-forward in that it required a one class program and a method that implemented a formula given. The comprehension questions required tracing through the code and explaining the output where both questions consisted of approximately 42 lines of code. As indicated in [6], the difficulty of assessing complexity of programming questions is a challenge for educators. For ease of reference sections of these questions are reproduced here. (Refer to appendix for the entire question.)

<u>*Question 4*</u>

For the following program you must make use of classes, constructors, objects and methods. Write a Java program that will calculate the distance between 2 points on the Cartesian plane and determine the equation of the line y= mx + c that passes through those 2 points.

The points (x1, y1) and (x2, y2) are input via the keyboard.

Distance = $\sqrt{(x2-x1)^2+(y2-y1)^2}$

Gradient m = $\dfrac{y2-y1}{x2-x1}$

*And the constant c = y1-m*x1*

<u>*Question 6*</u>

6.2 *If the line* ***"private int size =3"*** *was changed to* ***"private static int size = 3;"*** *what would the output be, given the same input as 10.1, and why?*
6.3 *If the line* ***"public void draw( )"*** *was changed to* ***"private void draw( )"****, will the main method compile successfully? Explain your answer.*

The marks obtained for the questions 4 and 6 of the examination have been extracted from the data of the pre-service teachers' examination. The following table, Table 4, summarizes the means and standard deviations of the surveyed students on the two questions and on the final mark in programming. All marks are unscaled (raw) and are expressed as percentages:

Table 4: Descriptive Statistics for pre-service students' assessments

	N	Mean	Std. Deviation
Q4%	12	68.58	31.064
Q6%	12	46.58	29.944

A pattern similar to that of the in-service teachers has emerged. Students performed better in the question that required solving a problem, rather than the comprehension required of question 6. Students seem to find the reading and understanding of code more difficult, as opposed to writing a complete program. Basic OO concepts were tested for understanding. The difference between class wide variables and the value of object attributes were confusing for students. According to Bloom's taxonomy, creating a program is a higher level question than reading and understanding. A possible explanation is that marks are given for method, even if the problem was not solved completely. It could be easy to accumulate marks in this way without really solving the problem.
The Pearson's correlation matrix is shown below, in Table 5.

Table 5: Pearson's correlation

	Q4%	*Q6%*
Q4%	1	
Q6%	0.01172	1

For the above statistical test the null hypothesis is:
H_0: A student doing well in Q4 (writing a program to solve a problem) does not have a positive correlation with the student doing well in Q6 (comprehension type question).
The alternative hypothesis is:
H_a: A student doing well in Q4 does have a positive correlation with the student doing well in Q6.

From Table 5, it is clear that the null hypothesis is accepted. A correlation of 0.01172 between Q4 and Q6 indicates that they might be almost unrelated as the correlation is close to 0. The non-zero association might mean that although there is no positive or negative relationship between the two, they might still appear to be associated. In other words writing a program to solve a problem should imply an understanding of program code. However the reverse may not necessarily be true.

Possible explanations for the differences in performances in questions 4 (generation type) and 6 (comprehension type) are:

- Students may have been facing a learning problem, and
- the instructor may have given the students an example to study or solve that was a complete solution to a similar problem to Q4.
- It is also possible for a student with a partial understanding to be able to write correct code up to a certain level, without completely understanding the code. Marks can therefore be accumulated as they are allocated for method. The marking scheme showed the break. down of marks for each aspect of coding. Therefore, it would appear that an assessment that involves generating a program may not be a true reflection of students' competence in programming. The assessment techniques used encouraged accumulation of marks for method. Alternatively, it could mean that not much problem solving was required for Q4.
- Computer programming is taught in the context of a multilingual society, where English is the second language. The constant translation of information between languages can affect students at the surface level. Hence reading code in a programming language may add to the complexity of programming and affect their performance.

Alternatively, it appears that instructors are teaching with a procedural bias and therefore understanding code that uses objects and classes even at an introductory level poses difficulties for the students. The code generation questions appear to be a one class program that has similar characteristics to that of procedural programming. Assessment in terms of program generation appears to have a procedural bias and may therefore not be done in terms of OOP principles. Hence teaching and assessment may not correspond to each other.

5. SUMMARY

In this study, the performance in certain questions of the assessment has been examined in depth. Important trends were found. Firstly, in-service teachers with prior programming experience performed better than those without prior programming experience. This is an obvious notion. Secondly, questions that required an understanding of the program execution (Q5) were more poorly answered than those that required writing a simple one-class program (Q2). Stated differently, the findings suggest that generating code to solve a problem is more easily accomplished than comprehension of code. The implication of this result is that understanding of memory diagrams, or rather representation of programs in memory for the object-oriented programs, were poor and writing a simple one-class program that may be procedural in nature is more easily accomplished. Similarly, it was found that, for pre-service teachers, questions involving understanding and tracing code (Q6) were more poorly answered than those that required solving a problem (Q4 that required generating code). The result has similar implications as the result for the in-service teachers. Moreover, on close examination of the assessment questions it was found that some questions had a procedural bias. This may be indicative of the throwback to the procedural way of thinking. Teacher educators need to develop an approach to teach OOP so that students understand and realize that learning to program is more than learning a programming language. Marking strategies that allow marks to be accumulated for method without completely solving the problem may give students a false sense of achievement that they can write a program. The accumulation of marks accounts for the better performance in code generation questions than in code comprehension questions. An important note to remember is that while the two groups of students were based in different institutions, taught by two different instructors and wrote different examinations, the findings appear to be similar. One is inclined to believe that even experienced instructors in procedural programming need guidance in the approach to the teaching and assessment of OOP.

While these results cannot be generalized, they do have implications for teaching, which is discussed in the next section.

6. IMPLICATION FOR TEACHING

Knowing that teachers teach as they were taught [12] (even at a subconscious level), it became clear that if the object-oriented approach to programming is not infused in instruction during their practicum experiences, pre-service teachers will not graduate with the ability to create true object-oriented programs in the learning environment; and this cycle of teaching the way one was taught will be perpetuated. Invariably, the approach to programming in introductory courses is dependent on, and influenced by, the instructor's approach and the instructor, in turn, is influenced by his/her past programming and learning experience. If the goal is to learn OOP then teachers should use appropriate teaching and assessment strategies to teach OOP (i.e. emphasis should be on identifying and creating classes and objects first) and avoid using a procedural approach to teach programming, even if students undergo a longer learning curve before they become competent programmers. Teaching object-oriented programming is more than teaching object-oriented programming languages even at the introductory level.

In the light of the procedural background of the in-service teachers and other instructors of programming, it is particularly pertinent that both pre- and in-service teachers be taught programming with the OO approach, while embracing aspects of procedural programming that are relevant to programming in general as is suggested by Lewis [15]. The national change in curriculum in IT to an OOP language makes this approach an obligation for teacher training in computer science education. Thus to support the transition a broader approach to exemplars for teaching would be required than simply a set of exercises. The goal should therefore be to develop a repository of tasks that would be appropriate to teach OOP. The results also suggest a clear need for a programme of ongoing teacher development. The underlying philosophy of different code organization and the study of other approaches will certainly offer insights into teaching methodology.

REFERENCES

[1] Bennedsen, J., & Caspersen, M. E. (2004). Teaching Object-Oriented Programming- Towards teaching a Systematic programming process. 18th European Conference on Object-Oriented Programming (ECOOP). June 14-18, Oslo, Norway.

[2] Bergin, J., & Winder, R. (2000). Understanding Object-Oriented Programming. Retrieved: May, 2010. http://csis.pace.edu/~bergin/patterns/ppoop.html.

[3] Bergin, S., & Reilly, R. (2005). Programming: Factors that Influence Success. SIGCSE '05 February 23-27. St.Louis, Missouri, USA.

[4] Brandt, C. (2008). Integrating feedback and reflection in Teacher preparation. ELT Journal, 62(1): 37-46.

[5] Cantwell-Wilson, B. and Shrock, S. (2001). Contributing to success in an introductory computer science course: a study of twelve factors. In Proceedings of the thirty-second SIGCSE technical symposium on Computer Science Education, pp. 184-188.

[6] Carbone, A. (2007). Principles for Designing Programming Tasks: How task characteristics influence student learning of programming. Unpublished PhD thesis. Monash University: Melbourne.

[7] Carr, S.C. (2002). Assessing learning processes. Intervention in School & Clinic, 37(3), 156, 7p, 2 charts.

[8] Department of Education. (2003). National Curriculum Statement Grades 10 – 12 (General), Information Technology. Retrieved February 2010 http://www.education.gov.za/Curriculum/substatements/ InformationTechnology.pdf

[9] Détienne, F. (1990). Expert programming knowledge: A schema based approach. In J.M. Hoc, T.R.G. Green, R. Samurcay, & D.J. Gillmore (Eds.), Psychology of programming (pp. 205-222). London: Academic Press.

[10] Gries, P. & Gries, D. (2002). Frames and Folders: a teachable Memory model for Java.. Journal of Computing Small Colleges, 17(6): 182-196.

[11] Griffiths, R., Holland, S. & Edwards, M. (2007). Sense before syntax: a path to a deeper understanding of objects. ITALICS, 6(4): 125-144.

[12] Gupta, R. (2004). Old habits die hard: literacy practice of pre-service teachers. Journal of Education for Teaching, 30(1):67-78.

[13] Gal-Ezer, J.,Vilner, T. & Zur, E. (2009). Has the paradigm shift in CS1 a harmful effect on data structures courses: a case study. SIGCSE Bulletin 41(1): 126-130.

[14] Kölling, M. (1999). The Problem of teaching Object-Oriented Programming, Part 1: Languages. Journal of Object-Oriented Programming, 11(8):8-15.

[15] Lewis, J. (2000). Myths about Object-Orientation

and its Pedagogy. SIGCS 2000 3/00 Austin TX, USA.

[16] Lister, R., Berglund, A., Clear, T., Bergin, J., Garvin-Doxas, K., Hanks, B.,et al. (2006). Research Perspectives on the Objects-Early Debate. SIGGSE Bulletin, 38(4): 146-165.

[17] Ritzhaupt, A. D. & Zucker, R. J. (2006). Teaching

Object- Oriented Programming Concepts Using Visual Basic .NET. Journal of Information Systems Education,

[18] Robins, A., Rountree, J. & Rountree, N. (2003). Learning and Teaching Programming: A Review and Discussion. Computer Science Education, 13(2): 137- 172.

[19] Sanders, K. & Thomas, L. (2007). Checklists for grading object-oriented CS1 programs: concepts and misconceptions. Proceedings of the 12th conference on Innovation and Technology in computer science education (ITiCSE). Dundee, Scotland, ACM.

[20] Simon, Lopez, M., Sutton, K., & Clear, T. (2009). Surely We Must Learn to Read before We Learn to Write! In M. Hamilton & T. Clear (Eds), Conferences in Research and Practice in Information Technology, 95: 165-170. Wellington, New Zealand: ACS.

[21] Stroustrup, B (1994). The Design and Evolution of C++. Addison- Wesley, Reading MA.

[22] Turner, S. A., Qunitana-Castillo, R., Pérez-Quiňoes, M.A. & Edwards, S.H. (2008). Misunderstandings about Object-Oriented Design: Experiences Using Code Reviews. (2008). SIGCSE Bulletin 40(1): 97-101.

[23] Wiedenbeck, S. & Ramalingam, V. (1999). Novice comprehension of small programs written in the procedural and object-oriented styles. International journal Human-Computer Studies 51:71-87.

APPENDIX

Exam paper –In-service

Question 2

*Write a program that calculates and displays the weekly salary for an employee who earns R175 an hour, works 40 regular hours, 13 overtime hours, and earns time and one-half (wage*1.5) for overtime hours worked. Create a separate method to do the calculation and return the result to be displayed. Save the program as Salary.java.*
Marks will be given for showing planning in the form of comments in your program and method structure.

Question 5

Consider the following class Date *and class* UseDate *written by a student:*

```
public class Date
{
private int year;
private int month;
private int day;
public Date (int y, int m, int d)
{
            year = y;
            month = m;
            day = d;
 }

public boolean isLeap()
 {
   int year = 2004;
   if (year % 4 == 0)
    return true;
   else
  return false;
  }
 }

import java.awt.*;
import hsa.Console ;

public class UseDate
{
  static Console c;          // The output console
  public static void main (String [] args)
  {
   c = new Console ();
   c.print ("Enter year, month, day (separated by
                                          spaces) :");
   int year = c.readInt();
   int month = c.readInt();
   int day = c.readInt();
   Date userDate = new Date (year, month, day);
   if (userDate.isLeap())
   c.println("Is leap year");
  else
   c.println("Is not leap year");
  } // main method
}    // The "UseDate" class.
```

Depending on the year entered by the user, the program should display "Is leap year" or "Is not a leap year". The problem is that no matter what the user types in, the program always displays "Is leap year".

5.1 Explain why the program is not working as it should and what can be done to fix the problem.
5.2 Describe in detail how you could use variable box diagrams and arrows (if need be) to explain this problem to a class of learners. Your diagram must be accompanied by a textual description. Explained what happens with and without the change as described.

5.3 Write a second constructor for the Date class which takes a single parameter of type String: a date in the form dd/mm/yyyy (for example, "24/09/1998"). Your constructor should extract the day, month and year parts from this string parameter and use them to initialise the data members of the Date class. You are not required to do any checking for invalid values. (Hint: to convert a String to an integer, use the method Integer.parseInt,).

Exam paper- Pre-service

Question 4

For the following program you must make use of classes, constructors, objects and methods. Write a Java program that will calculate the distance between 2 points on the Cartesian plane and determine the equation of the line y=mx + c that passes through those 2 points.

The points (x1, y1) and (x2, y2) are input via the keyboard.

Distance = $\sqrt{(x2-x1)^2+(y2-y1)^2}$

Gradient m = $\dfrac{y2-y1}{x2-x1}$

*And the constant c = y1-m*x1*

Question 6

```
public class LineTestExam

{
        private int size = 3;
        private char pattern = '*';

public void setSize (int s)
{
         If (s>=0)
          Size = s;
}
public void setPattern (char p)
{
          for (int i = 1; i<= size; i++)
            {
               for (int x =1; x<= i; x++)
                   System.out.print(pattern);
             System.out.println( );
              }
 }
}

import Utilities.*;
public class LineTestE
{
Public static void main (String [] args)
{
    LineTestExam  line1 = new LineTestExam();
    Line1.draw( );

    System.out.println ("Enter the size of line you want");
    Int num = Keyboard.getInt( );
    System.out.println("Enter the pattern of line you
want");
    Char pat = Keyboard.getChar( );
    Line1.setSize (num);
    Line1.setPattern(pat);
    Line1.draw( );
    LineTestExam line2 = new LineTestExam ( );
    Line2.setSize(5);
    Line2.draw ( );
    Line1.draw( );
  }
}
```

6.1 Trace through the program above and give the exact output when the main method is executed. Use as input; 4 for variable num and "%" for variable pat.

6.2 If the line ***"private int size =3"*** *was changed to* ***"private static int size = 3;"*** *what would the output be, given the same input as 10.1, and why?*

6.3 If the line ***"public void draw( )"*** *was changed to* ***"private void draw( )"****, will the main method compile successfully? Explain your answer.*