Using the Same Problem with Different Techniques in Programming Assignments: An Empirical Study of its Effectiveness

Michael Newby
Department of Information Systems & Decision Sciences
California State University Fullerton
Fullerton, California 92834, USA
mnewby@fullerton.edu

ThuyUyen H. Nguyen
Business Analysis, Systems and Information Management, Newcastle Business School
Northumbria University
Newcastle upon Tyne NE1 8ST, United Kingdom
thuyuyen.nguyen@northumbria.ac.uk

ABSTRACT

This paper examines the effectiveness of a technique that first appeared as a Teaching Tip in the Journal of Information Systems Education. In this approach the same problem is used in every programming assignment within a course, but the students are required to use different programming techniques. This approach was used in an intermediate C++ course. The assessment for the course consisted of four assignments and two examinations, one mid-term and one final. The first two assignments deal with basic C++ programming and functions, and the other two with classes and inheritance. The mid-term covers the basics of programming, including functions, and the final focuses on the use of classes and inheritance. The performance of students in the course was measured in the two semesters before and in the two semesters after introducing the use of the same problem. This was done by collecting the student scores for the assignments and examinations. Statistical analysis showed that there was a significant difference in the means of the scores for the last two assignments and the final before and after introducing the use of the same assignment problem, but no significant differences in the means of the scores for the first two assignments and the mid-term. This would indicate that using the same problem for assignments in programming classes could improve student learning by allowing students to focus on the technique, such as inheritance, rather than having first to understand new program requirements. This approach also has the advantage from an instructor's viewpoint, in that it will reduce the amount of time spent specifying assignments and the time spent in grading as well.

Keywords: Programming Assignments, Assessment, Effective Instruction, Student Learning

1. INTRODUCTION

Computer programming, in its modern form, has been around for over 60 years, and became a topic taught in universities in the 1960s. Since that time various techniques and tools have been developed to assist in teaching programming (Powers et al., 2006). However, there seems to be no agreement either on what we do when we teach programming (Blackwell, 2002), or the best way to do so (Fincher, 1999). Indeed, some authors argue that programming cannot be taught, with some students having aptitude and some not (Dehnadi and Bornat, 2006). For many years, the computer industry recognized that there was a disparity in the skill level of programmers, from novice to expert, with capability differences in terms of productivity including both development and debugging being a factor of 5 or greater (McConnell, 1998). Not surprisingly, most instructors would see the same differences in their students, between those who can program, and those who struggle. In these classes, students may be classified as those who do very well, those who cope, and those who do not understand the material. The problem with this situation is that teaching such a class is difficult because those who understand think the class does not cover enough material, whereas the students at the other end of the skill range believe the instructor goes far too quickly. Learning to program is hard, and so is teaching it (Dehnadi and Bornat, 2006). Students must be capable of abstraction, which is why many believe learning to program helps with general thinking skills (Rinard, 2008; van Roy et al., 2003).

This paper examines a technique suggested as a Teaching Tip (Newby and Nguyen, 2007). The approach is based on the Applied Apprenticeship method (Astrachan and Reed, 1995), where students are given well designed program code, then given similar problems to do on their own or in groups and encouraged to apply similar techniques (functions, classes, inheritance), even to the point of utilizing the instructor's code if this is appropriate. This is similar to real-life situations, where programmers are expected to work on someone else's code and also to work in a group (Fowler, 2004).

Where the experiential approach (Newby and Nguyen, 2007) extends the Applied Apprenticeship approach is that for all assignments, the same problem is used, but employing different programming techniques. Using the same problem for assignments has some similarities with the application-based approach adopted by Astrachan, Smith and Wilkes (1997) when teaching a data structures course, and to the case study approach to teaching programming advocated by Linn and Clancy (1992). In the course described in this paper, the different techniques used for the assignments were direct in-line code, function modularity, class modularity and inheritance. A sample assignment specification is given in the Appendix. Before adopting the same problem approach for assignments, each of the four programming assignments used a different problem. This was time-consuming for the instructor to come up with new problems, and time-consuming for the students to understand the requirements of each problem. When different problems were used, it was observed that many students did not seem to understand the purpose behind each technique and its use. This applied particularly to the concept of classes. The new approach of using the same problem appeared to improve student understanding and use of classes and inheritance. However, this was a purely subjective observation. Fortunately, student scores were available for assignments and examinations for some courses that used different problems for the assignments, and others that used the same problem. From these, it was possible to address the research question:

> "Does using the same problem in all assignments improve student understanding of the different methods and concepts in a programming course?"

This paper describes the course, the assessable work, the sample and the methodology, and discusses the findings.

2. APPROACHES TO TEACHING PROGRAMMING

There are a number of paradigms for the development of programs (Fincher, 1999), and an instructor chooses a paradigm and a language that satisfies their course requirements. For example, an instructor may teach a programming course by introducing objects early, or deal with the functional aspects first and then introduce object orientation afterwards. There is a fundamental difference between a program written using functions and one written using objects and classes. The difference is easily seen by looking at possible underlying diagrammatic representations of the programs. With a functional approach to programming, the underlying diagram is a structured flowchart, and a structure or hierarchy chart (Farrell, 2008). With an object-oriented approach, the diagrams are class diagrams, object diagrams, use case diagrams, etc. (Fowler, 2004). This means that to understand these two different abstractions, students must construct two different conceptual models.

The importance of the approach taken to teaching programming is highlighted by the observation that programming practices used by students in college are often carried over to their professional careers (McAndrews, 2000, p. 9), so if they are just interested in getting programs to work without fully understanding the concepts, that is what they are likely to do when they work in industry.

Most programming courses consist of lectures and laboratory classes, the lectures being used to introduce new material and the laboratory classes to provide students with the opportunity of practicing using this material in a controlled environment. Laboratory classes are important because programming is a skill and cannot be learned simply by reading a book but needs practice in order for it to be acquired (Azemi, 1995). This skill must be mastered before further progress can be achieved. Laboratories are a major component of learning in a programming course, and it has been demonstrated that a computer laboratory environment can affect learning and achievement (Newby and Fisher, 2000; Newby, 2002).

In terms of assessing student learning, there are a number of possible components including written examinations, laboratory examinations, structured laboratory exercises, and projects or assignments (Chamillard and Braun, 2000; Barros et al., 2003). Of these, both laboratory exercises and assignments play a dual role, one for assessment and the other for learning, although most students only see the former. The characteristics of laboratory exercises are that they are very specific to a topic, and take a short time to complete, usually one or two hours at the most. On the other hand, assignments are more complex, take much longer, and although they have specific goals, these goals are much broader than those for laboratory exercises.

For an assignment to be useful in both of its roles, it must be well specified, realistic, yet able to be completed, from a developer's perspective, in a relatively short period of time, measured in weeks. The time constraint issue applies to other disciplines, but is particularly important in Information Systems, where instructors strive to make the problems and cases realistic and relevant (Cappel and Schwager, 2002). From the point of view of learning, any skills gained by the student in completing one assignment must be able to be carried over into other assignments, so assignments must have specific goals, and must require the use of specific techniques. There is one further complicating factor, stemming from the time constraint placed on completing assignments, and that is that many students will do anything to get the program to work. Their focus is on correctness, and they will use any techniques to achieve it.

Instructors of programming courses seek ideas that may improve the effectiveness of their teaching, and many of these are very innovative (Benander and Benander, 2008; Miliszewska and Tan, 2007; Dunican, 2002). Journals covering education in Information Systems and Computer Science usually have a section on Teaching Tips. These

sections allow instructors to share their experience and approaches that they have found useful. However, there are few articles that attempt to measure whether a particular approach or technique actually improves student learning, although there are some exceptions (Al-Imamy, Alizadeh and Nour, 2006).

3. THE COURSE

The course involved in this change of teaching technique is an intermediate level one in C++. It is an option in the Information Systems major of a Bachelor of Business Administration program. The pre-requisite is an introductory course in programming in Visual Basic, which means that, for the overwhelming majority of students, this course is their introduction to the programming language C++. It is a single semester course over sixteen weeks, including finals, and, in that time, students have to learn and master the subject matter from basic skills in the language to the use of inheritance.

The main components of assessment for the course are four programming assignments and two examinations, one mid-term and one final. In the assignments, students develop an application which has been specified for them. In the first one, they write it using in-line code; in the second, they must use functions where appropriate; in the third, they use a single class, and in the last assignment, inheritance and polymorphism must be used. The assignments are graded according to a rubric that emphasizes structure and style, rather than correctness, even though the instructors recognize that correctness is the most important characteristic of production software. The reason for focusing on style and structure is that these characteristics of software are widely accepted as making a major contribution to correctness (Dromey, 1995).

The examinations are written, and do not involve use of any C++ environment. Students are required to write short sections of code, functions, and classes, either from scratch or derived from another class or classes using inheritance. They are also required to demonstrate that they understand how to use functions, classes and polymorphism. The examinations test not only what the students have learned in the lectures, but also, what they learned from the programming assignments and laboratory exercises. The mid-term examination covers the fundamentals of C++ programming and the use of functions, and the final covers classes and inheritance. The reason for not allowing the use of an environment is to determine whether students really understand the material or whether they just use the environment to find out what a section of code does. In a way, these examinations have similarities with those for certification (e.g. Oracle, Microsoft, Sun Microsystems), in that candidates are not permitted to use an environment. The only difference is that the examinations in this course require short answers, whereas for certification, multiple choice questions are used.

4. METHODOLOGY

The sample consists of four classes taught by the authors, one as instructor, and the other as teaching assistant, over four semesters. These are identified as semesters 1, 2, 3 and 4. All students were undergraduate, either Junior or Senior, and the maximum class size in each case was 40. The total number of students in semesters 1 and 2 was 70, and in semesters 3 and 4, it was 60. From the instructor's subjective observations, the students in each group were comparable in ability.

In the first two semesters, each of the four assignments was based on a different problem, and in the second two semesters, they were based on the same problem. In all semesters, solutions to the programming assignments were published before the next assignment was posted, and students were told they could utilize any part of the code they found useful. After the last assignment was graded, a solution to that assignment was posted. All assignments were graded by one of the authors using the same rubric, and all examinations were graded by the other author to give consistency. The point allocation used when grading the assignments was the same from semester to semester. The rubric for grading the assignments was made available to the students, so they knew how their programs would be assessed. However, they did not know how the points were allocated, so could not use the rubric to maximize their score.

5. ANALYSIS

Table 1 shows the descriptive statistics for the percentage scores for each assignment and examination grouped as semesters 1 and 2, and semesters 3 and 4.

5.1 Differences within semesters

A paired-samples t-test was carried out to examine differences in the mean between Assignments #1 and #2, #2 and #3, #3 and #4, and the Mid-term and the Final. The purpose of this was to determine whether there are any differences in the students' performances on the assignments and tests within each class irrespective of teaching approach. This was done for semesters 1 and 2 combined, and for semesters 3 and 4 combined. The results for semesters 1 and 2 are presented in Table 2, and for semesters 3 and 4 in Table 3.

The paired samples t-test for scores in semesters 1 and 2 shows significant differences ($p < .001$) in the means for Assignment #3 and Assignment #2, and for the final and mid-term examinations. The score for Assignment #3 is significantly less than that for Assignment #2, as is the mid-term examination when compared with the final. The effect size measures the importance of an effect, and Cohen (1988) suggests that a size of greater than 0.1 is small, 0.3 medium, and 0.5 large.

From these, it may be seen that the effect size for the differences between Assignment #2 and Assignment #3, and between the Mid-term and the Final are large, accounting for more than 25% of the variance in each case. The effect size for the difference between Assignments #1 and #2 is small. For semesters 3 and 4, there are no significant differences in the means of the scores for the assignments or the examinations.

	Before Change (Semesters 1 & 2)				After Change (Semesters 3 & 4)			
	Mean	S.D.	Min	Max	Mean	S. D	Min	Max
Assignment #1	85.3	12.15	38.9	100.0	86.5	8.42	53.0	100.0
Assignment #2	88.4	7.66	63.3	100.0	87.4	11.30	56.5	100.0
Assignment #3	80.0	7.37	60.0	93.0	85.4	16.46	62.0	100.0
Assignment #4	79.5	8.30	60.0	90.0	86.5	12.78	58.0	100.0
Mid-Term	80.6	13.83	44.0	98.0	79.1	15.04	30.0	100.0
Final	71.2	14.00	40.5	96.0	78.3	11.27	54.5	99.0
	N=70				N=60			

Table 1: Descriptive Statistics of the Samples

	Before Change (Semesters 1 & 2)			
	Mean of Difference	t-value	p-value	Effect size
Assignment #2 – Assignment #1	3.08	1.72	0.090	0.206
Assignment #3 – Assignment #2	-8.31	-7.13***	0.000	0.654
Assignment #4 – Assignment #3	0.45	-0.41	0.683	0.049
Final – Mid-Term	-9.45	-6.61***	0.000	0.623

*** $p < .001$

Table 2: Comparison of Differences between Means of Assignments and Examinations within Semesters 1 and 2

	After Change (Semesters 3 & 4)			
	Mean of Difference	t-value	p-value	Effect size
Assignment #2 – Assignment #1	1.03	0.68	0.496	0.091
Assignment #3 – Assignment #2	-1.94	-0.90	0.373	0.118
Assignment #4 – Assignment #3	-0.36	-0.14	0.886	0.019
Final – Mid-Term	-0.77	-0.47	0.641	0.061

Table 3: Comparison of Differences between Means of Assignments and Examinations within Semesters 3 and 4

5.2 Differences between semesters

An independent samples t-test was carried out on each of the assignments and the exams, grouping the variable on semester, with one group being semesters 1 and 2, and the other group being semesters 3 and 4. The purpose of this was to see if there were any differences in the mean scores for each assignment and each examination before and after changing to using the same problem for the assignments. As Table 1 shows large differences in the standard deviation before and after the change for some of the variables, Levene's test for the homogeneity of variance was performed. This is reported in Table 4. The results of the independent samples t-test are presented in Table 5. In this table, each t-value is appropriate to whether or not equal variances are assumed.

	Before Change S.D.	After Change S.D.	F- value	df1	df2	p-value
Assignment #1	12.15	8.42	3.025	1	120	0.085
Assignment #2	7.66	11.30	9.087	1	120	0.003**
Assignment #3	7.37	16.46	13.008	1	120	0.000***
Assignment #4	8.30	12.78	4.712	1	120	0.032*
Mid-Term	13.83	15.04	0.026	1	120	0.873
Final	14.00	11.27	5.326	1	120	0.023*

* $p < .05$, ** $p < .01$, *** $p < .001$

Table 4: Levene's Test for Homogeneity of Variance Based on the Mean

	Before Change Mean	After Change Mean	t- value	p-value	df	Effect Size
Assignment #1	85.3	86.5	0.604	0.547	126	0.054
Assignment #2	88.4	87.4	-0.595	0.553	99.0	0.053
Assignment #3	80.0	85.4	2.294	0.023*	72.5	0.200
Assignment #4	79.5	86.5	3.781	0.000***	93.3	0.320
Mid-Term	80.6	79.1	-0.594	0.553	127	0.052
Final	71.2	78.3	3.583	0.000***	126.6	0.305

*$p < .05$, **$p < .01$, ***$p < .001$

Table 5: Comparison of Means of Assignments and Examinations between Semesters

Levene's test shows that the standard deviations for Assignments #2, #3 and #4 are significantly greater after the change than before. For the final examination, the standard deviation after the change is significantly less than before.

Examining the means of the scores from the assignments and the examinations shows that for Assignments #3 and #4, and the final examination, the means after introducing the use of the same problem are significantly greater than those before. For Assignments #1 and #2 and for the mid-term exam, there is no significance between the means. The effect size for the differences in the means for Assignment #4 and the final examination was medium, and for Assignment #3 small.

6. DISCUSSION OF RESULTS

As stated earlier, Assignment #1 uses in-line code, Assignment #2 uses functions, Assignment #3 uses classes, and Assignment #4 uses inheritance. In a similar way, the mid-term examination tests knowledge of in-line coding and functions, and the final tests knowledge of classes and inheritance.

The results from the paired samples t-test supports the subjective observation that students in semesters 1 and 2 were not performing as well on Assignments #3 and #4 as they were on Assignments #1 and #2. There is a similar result for the final exam and the mid-term. This would imply that these students did not have as good an understanding of classes as they did of functions. It is to be expected that they would have a better understanding of functions as they had already met them in the pre-requisite course, whereas the concept of classes was new to the overwhelming majority of students. After the change was made to using the same problem for all four assignments, there were no significant differences between the students' performances on each of the assignments. Admittedly, this is no doubt in part because they had available to them program code (written by the instructor) from the previous assignment, but they also had similar code provided to them in the semesters before the change was made. Another possible explanation is that the courses were taught better after the change than before, but this is doubtful as the instructor has been teaching C++ programming classes for over 20 years, as well as other programming languages such as Cobol, Java, Visual Basic and C#. His teaching style has not changed significantly in the last 10 years. One notable observation to be made is that there was also no significant difference between performance on the mid-term and the final after the change was made. The final covers only classes and inheritance, and, in the final, students are asked to write small sections of code for applications they have not seen before. Further, they are not allowed to use an environment to test if the code they supply actually works. From this it may be inferred that by using the same problem for assignments, the students would appear to understand classes and inheritance to the same extent that they understand functions.

The results from the independent samples t-test indicate that using the same problem could help students understand the use of classes and inheritance better. The scores for students in Assignments #3 and #4, and the final examination are significantly higher after we implemented using the same problem compared with using different problems, with the effect size for the final being medium. Again for Assignment #3 and #4, this could be explained to some extent by students having access to 'good' code for the previous assignment, but this does not explain the improved performance in the final examination. An interesting observation is that the effect size for Assignment #3 is small, but for Assignment #4 it is medium. This could imply that once students understand classes, the transition to understanding inheritance is easier.

Levene's test for the homogeneity of the variance shows there are significant differences in the standard deviations before and after the change for Assignments #3 and #4, and for the final examination. For Assignments #3 and #4, the minima before and after are similar, but the means are significantly greater after the change, and this could imply that the most able and moderately able students are helped most by this approach. For the final examination, the minimum after the change is considerably greater than before the change, the mean is significantly greater, but the standard deviation is significantly smaller. It would seem from these results that all students, even the less able, gain better understanding of classes and inheritance from this approach. This is possibly because when they are studying for the final, they have the same sample problem using different techniques making it easier for the student to compare and understand the differences in the techniques.

Overall, these results would support the contention that using the same problem for assignments in a programming course could improve student understanding of different techniques and how to use them.

7. LIMITATIONS OF THE STUDY

Although the findings of the study demonstrate improvements in student outcomes in terms of achievement in examinations and assignments, there are two limitations that must be addressed.

The first one is that the GPAs for the students were not collected, and this limits us in determining if there were any differences in ability between the student groups across the semesters. In the study, it was assumed that students in the four semesters were comparable in overall ability. The second limitation is that student feedback was not incorporated into the research to determine what the students thought of this approach. It could be suggested that students may suffer from assignment fatigue, and become bored by having to write yet another program for the same problem. Anecdotal evidence would indicate that this is not the case for the majority. In fact, the most common response from students is firstly, surprise that they are doing the same problem followed by relief when they realize they do not have to work out new requirements. However, as student opinions were not sought, it is impossible to be definitive about whether they thought the approach improved their learning experience.

8. CONCLUSION

In any programming course, there are a number of components that contribute to the learning, such as lectures and structured laboratories, and there are other components that contribute to the assessment, and these include assignments, laboratory tests and examinations. As assignments involve program development over a period of time, not only do they contribute to the assessment, they also contribute to learning. This paper examined how the use of the same problem for four programming assignments could improve student learning outcomes. The assignments involved basic in-line coding, using functions, using classes and using inheritance respectively. Before switching over to using the same problems, different problems had been used for the assignments.

The results of the analysis would imply that using the same problem for a series of assignments does improve a student's understanding of classes and inheritance. There could be a number of reasons why this approach is effective, but the main one would be that, in the later assignments, the students can focus on the technique (either classes or inheritance), rather than spending their time trying to understand the problem requirements. The authors are aware that in a real-life application, developers do need to understand requirements, but the intention of this course is to teach programming techniques within a short period of time, not to develop a production system.

Apart from the improvement in student learning that may be seen from this approach, there are other benefits from the instructor viewpoint. Firstly, there is a reduction in time spent thinking up new problems that are realistic but able to be completed in a relatively short period of time. Secondly, less time is spent on explaining the program requirements, and finally, less time is spent on grading them.

This paper demonstrates that using the same problem across a series of programming assignments facilitates learning, and requires less effort on the part of the instructor. This technique is not specific to any programming language, and the instructor has applied it to courses involving Java and Visual Basic, and found similar improvements in student learning. It is an approach that the authors would recommend to any teacher of programming.

9. REFERENCES

Al-Imamy, S., Alizadeh, J., and Nour, M.A. (2006). "On the development of a programming teaching tool: The effect of teaching by templates on the learning process." Journal of Information Technology Education, Vol. 5, pp. 271-283.

Astrachan, O. and Reed, D., (1995). "AAA and CS 1: The Applied Apprenticeship Approach to CS 1" Proceedings of 26th SIGCSE Technical Symposium on Computer Science Education, Nashville, Tennessee, pp. 1-5.

Astrachan, O., Smith, R. and Wilkes, J. (1997), "Application-based modules using apprentice learning for CS 2", ACM SIGCSE Bulletin, Vol. 29, No. 1, pp. 233-237.

Azemi, A. (1995), "Teaching computer programming courses in a computer laboratory environment." Proceedings of ASEE/IEEE Frontiers in Education Conference, November 1995, pp. 2a5.18-2a5.20.

Barros, J.P., Estevens, L., Dias, R., Pais, R., and Soeiro, E. (2003), "Using lab exams to ensure programming practice in an introductory programming course." ACM SIGCSE Bulletin, Vol. 35, No. 3, pp. 16- 20

Benander, A.C. and Benander, B.A., (2008), "Student monks – Teaching recursion in an IS or CS programming course Using the Tower of Hanoi." Journal of Information Systems Education, Vol. 19, No. 4, pp.455-468

Blackwell, A.F. (2002), "What is programming?" In J. Kuljis, L. Baldwin & R. Scoble (Eds), Proceedings from the 14th Workshop of the Psychology of Programming Interest Group, Brunel University, June 2002, pp. 204-218.

Cappel, J.J. and Schwager, P.H. (2002), "Writing IS teaching cases: Guidelines for JISE submission." Journal of Information Systems Education, Vol. 13, No. 4, pp. 287-293

Chamillard, A.T. and Braun, K.A. (2000), "Evaluating programming ability in an introductory Computer Science course." Proceedings of the 31st SIGCSE technical symposium on Computer science education, pp. 212-216.

Cohen, J. (1988), Statistical power analysis for the behavioral sciences. Second edition. Academic Press, New York.

Dehnadi, S. and Bornat, R. (2006), "The camel has two humps." Retrieved November 18, 2009 from www.cs.mdx.ac.uk/research/PhDArea/**saeed**/paper1.pdf

Dromey, G.R. (1995), "A model for software product quality." IEEE Transactions on Software Engineering, Vol. 21, No. 2, pp. 146-162

Dunican, E. (2002), "Making the analogy: Alternative delivery techniques for first year programming courses." In J. Kuljis, L. Baldwin & R. Scoble (Eds), Proceedings from the 14th Workshop of the Psychology of Programming Interest Group, Brunel University, June 2002 , pp. 89-99.

Farrell, J. (2008), Programming logic and design. Fifth edition. Course Technology, Boston.

Fincher, S., (1999), "What are we doing when we teach programming?" Proceedings of 29th ASEE/IEEE Frontiers in Education Conference, San Juan, Puerto Rico, pp. 12a41-5.

Fowler, M. (2004), UML distilled: A brief guide to the standard object modeling language. Third Edition. Addison Wesley, Boston.

Linn, M.C. and Clancy, M.J. (1992), "The case for case studies of programming problems." Communications of the ACM, Vol. 35, No. 3, pp. 121-132

McAndrews, D. R. (2000), "The Team Software Process (TSP): An overview and preliminary results of using disciplined practices," Technical Report CMU/SEI-2000-TR-015, ESC-TR-2000-105. Carnegie-Mellon University, Software Engineering Institute.

McConnell, S. (1998), "Dealing with problem programmers." IEEE Software, Vol. 15, No. 2, March/April 1998, pp. 126-128

Miliszewska, I. and Tan, G. (2007). "Befriending computer programming: A proposed approach to teaching introductory programming." Issues in Informing Science and Information Technology, Vol. 4, pp. 277-289

Newby, M. (2002), "An empirical study of the learning environments of open and closed computer laboratories." Journal of Information Systems Education, Vol.13, No.4, pp.303 – 314

Newby, M. and Fisher, D.L. (2000), "A model of the relationship between computer laboratory environment and student outcomes in university courses." Learning Environments Research, Vol. 3, No. 1, pp.51-56

Newby, M. and Nguyen, T. (2007), "Using the same problem with different techniques in programming assignments." Journal of Information Systems Education, Vol.18, No. 3, pp.279-282

Powers, K., Gross, P., Cooper, S., McNally, M., Goldman, K.J., Proulx, V., and Carlisle, M. (2006), "Tools for teaching introductory programming: what works?" Proceedings of the 37th SIGCSE technical symposium on Computer Science Education 2006, pp. 560-561. ACM, New York.

Rinard, M. (2008), "Using programming language concepts to teach general thinking skills." Retrieved October 21, 2009 from people.csail.mit.edu/rinard//paper/wowcs08.pdf

van Roy, P., Armstrong, J., Flatt, M., and Magnusson, B. (2003), "The role of language paradigms in teaching programming." Proceedings of the 34th SIGCSE Technical Symposium on Computer Science Education, pp. 269-270.

AUTHOR BIOGRAPHIES

Michael Newby is a lecturer in the Department of Information Systems & Decision Sciences at California State University, Fullerton (CSUF). He received his B.Sc. from the University of London, U.K, his M.Sc. from the University of Bradford, U.K. and his Ph.D. from Curtin University in Western Australia. His research interests include computer learning environments, and improving student learning in programming courses.

ThuyUyen H. Nguyen is a senior lecturer in the Business Analysis, Systems and Information Management subject group at Newcastle Business School, Northumbria University. She received her BA and MS from California State University, Fullerton, and her PhD from the University of Teesside, UK. Her major research interests include Information Systems & Customer Relationship Management in SMEs. She is also interested in researching into how learning outcomes in Information Systems courses may be improved.

APPENDIX - TYPICAL PROBLEM SPECIFICATION

A travel company provides tour packages for its customers to three regions Europe, Asia, and Australia. To make it easier for customers, the company has adopted a simple pricing formula. You are to write a program to calculate the cost of each trip, based on which package the customer buys, whether airfare is included, and how far in advance the package is booked.

The basic land-only tour costs for different regions are:

Europe	$1950.00
Asia	$2250.00
Australia	$2550.00

The price discount based on advance booking is:

90 days or more	15%
between 30 and 90 days	5%
less than 30 days	0%

The airfare is charged separately from the land-only tour, and is not discountable. For those customers who wish to include the airfare in the package, it is calculated as followed:

To Europe:	60% of the land-only tour price
To Asia:	75% of the land-only tour price
To Australia:	90% of the land-only tour price

For each customer, you are to input the following:

Booking Number
Customer Name
Travel Region (as a character)
Number of days booked in advance (as an integer)
Whether airfare is included (as a character 'Y' or 'N')

The tour region is to be represented as a single character as follows:

Europe	'E'
Asia	'A'
Australia	'U'

After doing so, you are then to output for each customer the following values:

Booking Number
Customer Name
Tour region (as a string)
Cost of land tour-only before discount
Discount amount
Cost of land-only tour after discount (Cost of land-only tour before discount – Discount amount)
Airfare included (a string that says whether airfare is included or not)
Cost of airfare
Cost of the trip (Cost of land-only tour after discount + Cost of the Airfare)

Customer's information should be processed repeatedly until the null string is input for the Booking Number, at which point the following totals should be displayed:

Number of customers
Total Discount for all customers
Total Cost for all customers

The program should then terminate.

All input data should be validated and suitable error messages produced

Assignment #1: Write the program as in-line code without writing your own functions

Assignment #2: Write the program using appropriate functions

Assignment #3: Write the program using a class for the tour package, so that all functionality for the tour package is within the class.

Assignment #4: Write the program using an abstract class for tour package, with three derived classes, one for Europe tours, one for Asia tours and one for Australia tours.

Information Systems & Computing
Academic Professionals

STATEMENT OF PEER REVIEW INTEGRITY

All papers published in the Journal of Information Systems Education have undergone rigorous peer review. This includes an initial editor screening and double-blind refereeing by three or more expert referees.

ISSN 1055-3096