Journal of Business & Leadership: Research, Practice, and Teaching (2005-2012)

Volume 7
Number 1 *Journal of Business & Leadership*

Article 5

1-1-2011

Objects-First Vs. Structures-First Approaches To 00 Programming Education: A Replication Study

Richard A. Johnson
Missouri State University

Duane R. Moses
Missouri State University

Journal of Business & Leadership: Research, Practice and Teaching
2011, Vol. 7, 37-41

OBJECTS-FIRST VS. STRUCTURES-FIRST APPROACHES TO OO PROGRAMMING EDUCATION: A REPLICATION STUDY

Richard A. Johnson, Missouri State University
Duane R. Moses, Missouri State University

One of the essential elements of a successful organization is information technology, which has as its basis effective and efficient software development. In turn the foundation of software development is computer programming. The last decade of computer programming education has been dominated by the object-oriented paradigm. While recent anecdotal accounts among computer science and computer information systems educators have often favored the objects-first approach to programming instruction (vis-à-vis the structures-first approach), very little empirical evidence has been offered. A field study by Johnson and Moses (2008) suggested that the objects-first approach is superior, but the experimental design was open to criticism. This replication study significantly improves upon the experimental design yielding results that indicate neither the objects-first or the structures-first approach is preferred. While an inconclusive result may seem unimportant, it does provide needed guidance to educators to make pedagogical decisions based on other perhaps more important factors to help ensure the students' success. On the positive side, the study does suggest that learning object-oriented programming is more difficult for novices than learning procedural programming, which is also important for programming educators.

Virtually all kinds of organizations are heavily dependent on information technology for survival and success, and information technology is predicated on the efficient and effective application of computer programming techniques. These programming techniques are usually first developed to a high degree within computer information systems (CIS) and computer science (CSC) curricula. In the past twenty years, virtually all introductory programming courses have shifted from the procedural approach to the object-oriented (OO) approach. Most beginning programming courses appear to be teaching Java , C++, or one of the Visual Studio .NET languages (Visual Basic, C#, or J#) as evidenced by the popularity of various computer programming texts. All of these programming languages are object-oriented, as contrasted with the purely procedural languages of Fortran, Pascal, COBOL, and C, which were highly popular prior to 1990.

The basis of any type of computer programming (OO or procedural) involves the three programming 'structures': (1) sequence (do A, do B, do C, …), (2) selection (if…else decisions), and (3) repetition (while or for loops). OO programming includes these structures but extends them by creating classes that serve as templates for instantiating software objects in computer memory. Objects, characterized by attributes (instance fields) and behaviors (instance methods), often represent entities in the real world (such as students, products, or airline reservations). While learning the basics of procedural programming with structures (sequence, selection, and repetition) is often difficult for most beginning students, it is the general consensus that learning OO programming concepts and techniques may be even more challenging (Sheetz, et al., 1997; Robins, et al., 2003).

Therefore, one of the most relevant questions regarding programming education should be whether to teach procedural programming concepts and techniques first, followed by OO programming concepts and techniques, or vice versa. Many authors claim it is better to follow an objects-first (OF) approach ('early objects') in order to ingrain within the student the ability to think in terms of objects (Thramboulidis, 2003; Ragonis & Ben-Ari, 2005). Although the OF approach may sound more plausible than a structures-first (SF) approach, there appears to be little empirical evidence to support the claim.

An original study by the authors of this paper (Johnson & Moses, 2008) was conducted to test the hypothesis that there is no difference in an OF and a SF approach to teaching OO programming to beginning students. The OF approach was used in one section of introductory Java programming by one instructor while the SF approach was used by another instructor in a different section of the same course. The student populations of the two sections were statistically identical, but the OF group significantly outperformed the SF group on identical final exams, which consisted of writing a challenging OO application. While great care was taken to teach the OF and SF sections in much the same way (similar combinations of lecture and lab) and grade the final exams similarly, each instructor had his own unique teaching style and grading technique, which may have contributed to the differences in student performance on the final exam.

The current study seeks to greatly reduce any threats to validity by having just one instructor teach two separate sections of introductory Java programming, one section following the OF approach and the other following the SF approach. This way the background and teaching style of the instructor is the same for both sections. Any bias in grading the final programming application was also eliminated by having the other author (an experienced OO programming instructor) grade the exams from the two sections (in random

order) without knowing which section (OF or SF) served as the source of the exams.

METHOD

Procedures

The research question driving this study is: What effect does teaching an OF approach (vis-à-vis a SF approach) have on the performance of novice programming students in an OO programming course? The hypothesis being tested is that there is no difference in the performance of novice programming students who are provided with an OF or a SF approach to OO programming.

To test this hypothesis, one of the authors of this study taught two sections of introductory OO programming (CIS 260), each with 24 students, at Missouri State University (MSU). The following steps were taken in the administration of these two sections:

1. The instructor used two different texts written by the same author (Gaddis, 2008a; Gaddis, 2008b), one using the term "Early Objects" in its title and the other using the term "Late Objects" in its title. The only significant difference between the two texts is the ordering of the chapters. Both texts have the same first two chapters. The Early Objects text presented objects and classes beginning with Chapter 3 (we call this the OF approach) while the other Late Objects text continued after Chapter 2 with decisions, loops, and methods before introducing classes and objects in Chapter 6 (e call this the SF approach). These two texts were designed specifically by Gaddis to support either an OF or a SF approach to teaching introductory OO programming using Java. The reading material, programming examples, and end-of-chapter problems throughout the texts are essentially the same, although located within different chapters.
2. Both sections were taught in the same lab setting using identical lecture/discussion/demonstration techniques by the same instructor. The instructor also assisted students with writing programs during specified lab time. No graduate assistants or other instructors were used in the delivery of these two sections.
3. Students in both sections took the same first exam covering Java programming basics (Chapters 1 and 2 in both Gaddis texts) and the same final exam consisting of writing complete Java code for a challenging OO application.
4. To avoid bias, the other author of this paper (not the instructor of the two sections) graded all final exam programs. The exams were completely randomized so that the grader did not know which students belonged to the OF section or the SF section.
5. The grading was performed in great detail for each exam. The evaluation was broken down into 17 specific programming tasks within each of two broad categories of OO programming (such as creating instance fields or instantiating objects) and structured programming (such as writing loop structures or writing methods).
6. Percentage scores on the final exam were determined for each student within the broad categories of OO programming and procedural programming based on the 17 tasks within each category. The means of these scores were compared for the OF an SF groups to test the null hypothesis of this study (i.e., no difference in the performance of the OF and SF groups).

Student Backgrounds and Demographics

Data about the students were collected during the first week of class to determine if both groups (OF and SF) had similar backgrounds and abilities. Students completed a survey to provide demographic data such as gender, age, college class, prior experience in programming, and motivation to learn computer programming. The college GPA and ACT scores (composite and math) were also collected for all students.

Performance Measures

The first exams in each section were identical, being based on the identical first two chapters in both Gaddis texts (Early Objects and Late Objects). These two chapters covered the very basics of Java programming (variables, data types, type casting, strings, etc.) with practically no emphasis in OO programming. This first exam, consisting of multiple choice questions and writing a short Java program, was scored by the same instructor for both the OF and SF sections.

After Chapter 2 in the Gaddis texts, the sequence of topics was totally different. A second exam was administered in both sections approximately half way through the course but the sections could not be compared due to the different topics covered.

After both sections completed their respective courses, a final exam was administered. Students were asked to write (using paper and pencil) a complete syntactically and logically correct OO application in the Java language. The authors believed it would be better to have the students write programs on paper instead of on a computer so that credit could be given for code that was close to correct. The authors understood that incorrect syntax or logic on the computer would result in early failure of the program to compile or run which would perhaps lead to undo frustration on the part of the student and less likelihood of writing a complete program.

Following is the fairly challenging programming problem on the final exam (the students had two hours to write the application):

https://scholars.fhsu.edu/jbl/vol7/iss1/5
DOI: 10.58809/FFIX4475

Write two Java programs, Book.java and BookApp.java. Use JOptionPane for input and output. The Book class has three instance fields: bookID (int), bookTitle (String), and bookPrice (double). The Book class has a static field, TAX_RATE = 0.05, which is a named constant, and a static field, count. The count should be increased by 1 every time the constructor is invoked. The Book class has two constructors: one has two parameters only (theId and theTitle) while the other has three parameters (theId, theTitle, and thePrice). The Book class has a method called calculateTax() which uses TAX_RATE and bookPrice to return the tax on a book. The Book class has a method called calculateFinalCost(), which calculates the final cost of the book to the customer (bookPrice + tax). The Book class has get and set methods for all instance fields. The BookApp class first declares all variables to be used in that class. Include a DecimalFormat object to format numbers to two decimal places with a dollar sign. The BookApp class has a while loop that allows the user to continue to create new book objects until the user chooses to stop. Within the while loop, the BookApp class gets input from the user for bookId, bookTitle, and bookPrice, creates a Book object, calculates the tax using the calculateTax() method and calculates the final cost using the calculateFinalCost() method. Within the while loop, the BookApp class displays, in a JOptionPane window, all the information about the book object created including the ID, title, price, tax, final cost, and number of books processed.

The researcher who graded these final exams (not the instructor of the sections) identified 17 key OO programming tasks (such as correct class declaration in the Book class, correct declaration of three private instance fields in the Book class, correct instantiation of a DecimalFormat object in the BookApp class, etc.) and 17 key procedural (or non-OO) programming tasks (such as correct definition of the calculateTax() method in the Book class, correct while loop in the BookApp class, correct conversion of String input to numeric data in the BookApp class, etc.). This instructor then carefully evaluated each exam based on these 34 criteria, deducting 0 points for correctly performing the task, 0.5 points for a minor error in performing the task, and 1.0 point for a major error in performing the task. The result was a percentage correct for the OO tasks and a percentage correct for the procedural (non-OO) tasks for each student within the OF and SF sections.

RESULTS

Table 1 shows various demographic data for the OF and SF groups. Table 2 presents the results from the first exam and the final exam. The mean scores for the first exam in the two sections (OF and SF) are compared. For the final exam the mean OO programming scores are compared and the mean non-OO programming scores are compared for each of the two sections (OF and SF) using t-tests. Finally, Table 3 compares the difference in scores for non-OO and OO programming by student within each of the two sections (OF and SF) using paired sample t-tests. Finally, Table 4 provides the means of the differences in non-OO vs. OO programming performance for the OF and SF sections.

TABLE 1

Demographic Data for the Objects-First (OF) and Structures-First (SF) Sections

		OF	SF	P value (two-tailed)	H_0: $\mu_{OF} = \mu_{SF}$ $\alpha = 0.10$
1	# Male	22	21		
2	# Female	2	3		
3	Total students	24	24		
4	Age	21.5	20.5	0.113	FTR H_0
5	Previous college GPA	3.09	3.01	0.325	FTR H_0
6	ACT score (composite)	24.8	25.0	0.858	FTR H_0
7	ACT score (math)	25.0	24.3	0.572	FTR H_0
8	Semesters of programming experience	1.29	1.25	0.933	FTR H_0
9	Course is important to my career	3.88*	3.75*	0.664	FTR H_0

*Used a Likert scale of 1-5 for strongly disagree to strongly agree

TABLE 2

Programming performance data for the objects-first (OF) and structures-first (SF) sections

		OF	SF	P value (two-tailed)	H_0: $\mu_{OF} = \mu_{SF}$ $\alpha = 0.10$
1	Exam 1 (beginning programming basics)	89.1%	88.8%	0.878	FTR H_0
2	Final Exam: OO tasks	73.4%	73.9%	0.926	FTR H_0
3	Final Exam: Non-OO tasks	83.3%	87.5%	0.271	FTR H_0

TABLE 3

Difference in programming performance by student, Non-OO score – OO score (using paired t-test) for OF and SF sections

		OF	SF	P value (two-tailed)	H_0: $\mu_{Non\text{-}OO} = \mu_{OO}$ $\alpha = 0.10$
1	Final Exam: Non-OO score – OO score	9.9%		<0.001	Reject H_0
2	Final Exam: Non-OO score – OO score		13.6%	<0.001	Reject H_0

TABLE 4

Comparison of mean differences in Non-OO and OO scores between OF and SF sections

		OF	SF	P value (two-tailed)	H_0: $\mu_{OF} = \mu_{SF}$ $\alpha = 0.10$
1	Final Exam: Non-OO score – OO score	9.9%	13.6%	0.246	FTR H_0

DISCUSSION

Table 1 demonstrates that demographically the OF and SF sections are virtually identical. However, the SF section is slightly younger (Line 4), almost but not quite statistically significant.

Table 2 provides strong evidence that the OF and SF sections had nearly equal basic programming skill (Line 1). Both sections had nearly the same prior experience in programming (about one semester), presumably either in high school or junior college. Both sections also demonstrated nearly equal abilities in OO programming (Line 2). However, the SF sections did score slightly higher on the non-OO programming tasks on the final exam, although not statistically significant. The most striking finding of Table 2 is that both sections scored higher on the non-OO programming tasks than the OO programming tasks. While some early practitioners claim that the OO approach is more natural and easier (Booch, 1994), most recent research suggests that OO programming is in fact more difficult than structured programming (Robins, et al., 2003).

The natural question from Table 2 is whether these differences in non-OO and OO scores for each section are indeed significant. Table 3 concludes that these differences are highly statistically significant. Regardless of the approach to OO programming (OF or SF), students perform much better on non-OO programming tasks. And these differences in non-OO vs. OO performance appear to be greater for the SF section. However, Table 4 shows that the difference in the two sections is not statistically significant. But there is the suggestion that students exposed to structures early perform slightly better on structured programming tasks at the end of the course. The major conclusion of this study is that an OF or SF approach to OO programming made essentially no difference in OO programming performance. This finding is important because it can free educators to explore other opportunities to improve programming education without necessarily being tied to an OF or SF approach.

CONCLUSION

Learning programming is not an easy task for the novice student. Learning OO programming is an even more daunting task (Robins, et al., 2003). This study compared the performance of two nearly identical groups of novice programming students. One group studied objects and

https://scholars.fhsu.edu/jbl/vol7/iss1/5
DOI: 10.58809/FFIX4475

classes very early in the semester (the objects-first, or OF, group) while the other group studied the basic programming structures (sequence, selection, and repetition) before objects and classes (the structures-first, or SF, group). Both groups took the same first exam (covering only the most basic Java programming tasks) before they diverged into either the OF or SF approaches. Then both groups took the same final exam which covered complete elementary OO development. The OF and SF groups were statistically identical in their performance on the first exam. Surprisingly, both groups also scored the same on the OO programming tasks on the final exam and statistically the same on the non-OO programming tasks on the final exam. However, both groups performed at a statistically higher level on structured programming tasks than object-oriented programming tasks, suggesting that OO programming is indeed more difficult regardless of the approach. These experimental results suggest that the approach taken to teaching OO programming does not affect programming performance, especially OO programming performance.

REFERENCES

Booch, G. (1994). *Object Oriented Analysis and Design With Applications*, 2nd ed. Redwood City, CA: Benjamin/Cummings.

Gaddis, T. (2008a). *Starting Out with Java: Early Objects*, 3rd ed. Boston, MA: Pearson.

Gaddis, T. (2008b). *Starting Out with Java: From Control Structures through Objects*, 3rd ed. Boston, MA: Pearson.

Johnson, R. & Moses, D. (2008). Objects-First vs. Structures-First Approaches to OO Programming Education: An Empirical Study. *Academy of Information and Management Sciences Journal, 11*, 95-102.

Ragonis, N. & Ben-Ari, M. (2005). A Long-Term Investigation of the Comprehension of OOP Concepts by Novices. *Computer Science Education, 15*, 203-221.

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

Sheetz, S., Irwin, G., Tegarden, D., Nelson, J., & Monarchi, D. (1997). Exploring the Difficulties of Learning Object-Oriented Techniques. *Journal of Management Information Systems, 14*, 103-131.

Thramboulidis, K., (2003). A Constructivism-Based Approach to Teach Object-Oriented Programming. *Journal of Informatics Education and Research, 5*, 1-14.

Richard A. Johnson is a Professor of Computer Information Systems at Missouri State University – Springfield where he has been teaching web application development and Java programming for the past fourteen years. Dr. Johnson received an MBA degree from Missouri State University, a Master of Engineering degree from North Carolina State University, and a Ph.D. in Business Administration from the University of Arkansas. He has been published in the Communications of the ACM and IEEE Transactions in Engineering Management and has authored several books on Java and Visual Basic programming.

Duane R. Moses is an Associate Professor of Computer Information Systems at Missouri State University – Springfield. He has been teaching Java programming, as well as, Management Information Systems in the Executive MBA program. In 2009, Dr. Moses taught Web Development at the Missouri State University branch campus in Dalian, China. Dr. Moses completed his BS and MS degrees at Oklahoma State University and his Ph.D. at the University of Missouri – Columbia. He has been published in the Academy of Information and Management Sciences Journal, Journal of Information Systems as well as the College Student Journal.