Which Introductory Programming Approach Is Most Suitable For Students: Procedural Or Visual Programming?

Chaker Eid, University of Bahamas, Bahamas
Richard Millham, University of Bahamas, Bahamas

ABSTRACT

In this paper, we discuss the visual programming approach to teaching introductory programming courses and then compare this approach with that of procedural programming. The involved cognitive levels of students, as beginning students are introduced to different types of programming concepts, are correlated to the learning processes of programming. Our hypothesis is that if beginning students are introduced to programming concepts by means of a console-based procedural programming approach, they perform better in subsequent visual programming higher level courses. The performance of two groups of students, one group who began with the console-based procedural programming approach and then advanced to a higher level visual programming course and the other group which began with a lower level visual programming course before proceeding to the same higher level visual programming course, is measured, analysed, and the results are reported, with statistical analysis, and correlated to the hypothesis.

Keywords: Procedural Programming; Visual Programming; Cognitive Development

INTRODUCTION

At present, visual programming is becoming increasingly popular as an introductory programming language. Students have a hands-on experience as they are able to manipulate visual concrete building blocks of a program, such as textboxes and buttons, and quickly produce an application. However, the question remains: is visual programming the best way to introduce beginning students to programming and are these students capable of mastering the complexities of visual programming in a one semester course of fourteen weeks? Through visual programming, do they obtain a grasp of the basic concepts of programming that is needed for them to advance to a higher-level programming class and to succeed in it? Our hypothesis is that a higher success rate in advanced visual programming classes is obtained when students are given an introductory programming course, using console-based procedural programming, where they gain a strong understanding of basic programming concepts such as procedures, variables, and loops without needing to learn the added complexities of a visual interface, et al than if the beginning student was given an introductory visual programming class, with the required learning of additional complexities and functionalities such as the integrated development environment (IDE), the procedural code-behind associated with visual controls, the visual interface, and the need to correctly map the interface to the solution requirements of the application problem (four levels of complexity), when they are introduced to visual programming immediately. In order to test our hypothesis, we studied the performance, in an advanced level programming class, of two groups of students – one group whose introductory programming class was a console-based procedural programming and the other group whose introductory programming class was introductory visual programming. The performance of those two different groups, in an advanced visual programming course, are measured, analysed, and correlated to the test group, who had console-based procedural programming as their introductory programming course, as compared to the control group whose introductory programming course was visual programming.

In order to discuss procedural and visual programming approaches, these approaches must be defined. Louden defines procedural programming as a sequential execution of instructions and the storage of values and the manipulation of these values, via variables, within the program (Louden, 1993). Visual programming can be defined as the manipulation of visual objects on a computer screen. This visual manipulation provides a concrete component which enables even those at the concrete operational level to handle this type of programming (Pendergast, 2006). In addition to visual objects on the screen, many visual programming languages, such as Microsoft Visual Basic, have procedures associated with each visual object, which are invoked by users, in order to perform their functionality. Consequently, besides manipulating virtual visual objects on the screen, students must know how to write the procedural code to implement the functionality associated with each of these objects and their events (event-handler procedure or "code-behind"). This procedural programming involves a formal level of cognition (Chiapetta 1976) In addition, visual programming involves the use of an integrated development environment (with additional functionalities of views, interactive debugging, etc) plus the need for students to be able to visualise user interface requirements into a well-ordered visual interface (Zak, 2005).

CHANGE IN PROGRAMMING LANGUAGE TEACHING

In order to understand the debate of the choice of introductory programming language, it is important to understand how the choice of introductory programming languages has changed. Up until 1990, the programming language of choice for IT students was COBOL, largely due to its popularity for use in business applications. However, from the 1990s, many computer science programs began to introduce students to procedural programming via the PASCAL programming language. The increasing use of personal computers, with the availability of PC-based procedural programming languages such as Turbo Pascal, reinforced this trend. By the late 1990s, the popularity of object-oriented programming languages along with the feasibility of visual programming made visual programming languages predominant (Pendergast, 2006) Now, educators are faced with the dilemma of what introductory programming language is best suited for their beginning students. In this paper, we argue that the best introductory programming language for beginning student is console-based, procedural programming, which stimulates higher level thinking amongst students without adding layers of additional complexity such as the IDE to confuse them.

BODY OF PAPER

Although most North American educational institutes have moved towards the visual programming approach of teaching introductory programming courses, we believe that it is more natural to reverse the course and introduce students to programming concepts via a console based procedural programming, in the same way that programming languages have evolved. This approach has the benefit of introducing students to elementary building bricks of programming one concept at a time and enabling them to move up on levels of complexities that require higher cognitive levels. For instance the concept of function or method or procedure is at a higher level that the concept of a variable or a loop and often encompasses one or several elementary concepts. Also the class is a higher logical concept than a function and functions make part of the body of a class. The analogy is that the brick is part of a wall and a wall is a part of a storey and the stories make up the building; our approach is to introduce the students to the brick first and then, using the brick as the first building block, move on higher up.

Our argument is partially based on the levels of cognitive processes that people adapt while learning a particular programming language. According to many researchers, the cognitive processes of people learning procedural versus visual programming are quite different and that these differences may account for how programming concepts are learned. In order to better understand how these cognitive processes relate to learning programming concepts, the cognitive development theory of one of the foremost researcher in this area, Piaget, must be understood. These states of Piaget's cognitive development theory correspond to the building blocks of programming from concrete (virtual visual blocks of the user interface) to formal (abstract logic embedded in procedural programming).

Piaget developed a cognitive development theory that consisted of three states of development: pre-operational, concrete, and formal operations (Piaget 1972 & Epstein, 1990). Pre-operational is the mental age from 2 years to 7 years. Concrete conceptualisation is the understanding of conservation of matter and

classification/generalisation (such as the conclusion that all cars are vehicles but not all vehicles are cars). However, Barker concludes that a person at the concrete level is unable to comprehend concepts such as mathematical ratios (Barker, 1983). The highest cognitive development level as defined by Piaget is the formal operations. Formal operations consist of the ability of a person to deal with abstractions, form hypotheses, systematically solve problems and engage in mental manipulations (Biehler and Snowman 1986). Lawson determined that a precondition to developing the ability for formal operations is an understanding of biconditional reasoning, "if and only if" logic. Biconditional reasoning plays an important role in procedural programming; if a person is unable to utilise biconditional reasoning and cannot make the move from concrete to logic/abstract thinking (Chiapetta 1976), procedural programming is difficult to learn (Becker, 1982). However, research has indicated that formal operations, such as abstract and logical thinking, may occur much later in some people or never occur (Griffiths 1973, Schwebel 1975, and Pallrand, 1979). In one study, only 17% of 7th graders, 23% of 8th graders, and 34% of 12th graders reach this formal operational stage (Renner, 1978). This study is buttressed by another study by Epstein that showed that while 20% of 13 year olds were at the formal operational stage, 78% were at the concrete operational stage and 2% remained at the preoperational stage of Piaget's cognitive development (Epstein, 1980). In regards to this paper, several studies indicated that a majority of adults, which include college students and professionals, fail at many formal operational tasks (Sund 1976 & Petrushka 1984). Griffiths and Schwebel have demonstrated that many college students do not obtain full formal operational thinking (Griffiths 1973 & Schwebel 1975).

Hudak argues that the greater abstract learning style, in the formal operational reasoning ability stage of Piaget's cognitive development theory, helps them learn procedural programming (Hudak and Anderson, 1990).

Although the manipulation of virtual visual objects on the user interface is intuitive at the concrete cognitive level, in order to implement the functionality associated with these objects (the procedural "code-behind"), the developer needs a higher, more abstract level of thinking. In order to develop the logic necessary for computer procedural programming successfully, several studies have demonstrated that formal operational reasoning ability is required (Fletcher 1984, Little 1984, Azzedine 1987, and Hudak 1990). Alustrum has demonstrated that beginning computer science students experience more problems understanding concepts that involve mathematical logic than other types of concepts, such as virtual manipulation of visual objects (Almstrum, 1994 & Losh, 1984). Cafolla has deduced that "some people of college age have difficulty learning procedural programming. This suggests that the cognitive skills needed to learn procedural programming develop later or perhaps never, in some" (Cafolla, 1987)

In order to tie in Piaget's cognitive development theory with that of dominant brain hemisphericity, the following table is provided. Montfort, through his study of brain interactions, found Computer Science and Mathematics students tended to be left brain dominant while music, art, oral communication, and journalism students had a tendency for right brain dominance (Montfort, 1990). Table 1 indicates the various programming paradigms and the cognitive development levels of Piaget associated with it.

Table 1: Programming Languages and Cognitive Development/Style Programming

	Piaget's Cognitive Development Levels Cognitive Style				
Paradigm	**Pre-Oper**	**Concrete**	**Pre-Formal**	**Formal**	**Formal (Hemisphericity)**
Procedural (COBOL, logic sequence)	Burnout	Burnout	Burnout	P & M	Left Brain
Object Oriented (C++, Java, concepts)	Burnout	Burnout	Burnout	P & M	Either hemisphere
Visual (Visual Basic	Burnout	Burnout	P & M	Bored	Either hemisphere
Script (HTML Web pages)	Burnout	P & M	Bored	Bored	Right Brain

(P & M; productive and motivated) (White, 2002)

The argument is that students, as part of the introductory programming course, should first learn console-based, procedural programming. This type of programming introduces students to the concepts of variables, iteration, selection, variable manipulation, and procedures with simple input and output displays. Students can focus on the basic concepts of programming without the confusing added complexities of classes, visual programming components, the functionalities of an integrated development environment, or the user interface design requirements. Once they master the basics of programming, they are better able to apply these concepts to additional

layers of complexity, such as a graphical user interface, in visual programming. Without such a background, students do not practice their higher level cognitive ability but focus on the concrete – develop attractive graphical user interfaces - yet, because they lack the background of procedural programming, are confused by programming that control the functioning of the interface (the "code behind") which closely resembles procedural programming, differing only in that the procedures may be invoked by events on the interface rather than called directly by the programmer (Chiapetti, 1976 & Piaget, 1972). An analogy to starting students with visual programming as opposed to procedural console based programming is to introduce pupils to algebraic equations solving before they master the operations on numbers (Piaget, 1972). The idea to introduce a beginning student to visual programming first might appear intuitive and easier for students to comprehend; this is, however, true only at the surface level since any response to any event involving a visual object involves an event-handling procedure that contains elements such as variables, logical statements et al. By introducing students to visual programming without having mastered elementary concepts contained within the procedure we are, in fact, introducing them to programming at three levels of complexity and then try to bring them down to the first level of complexity. This introduction to multiple levels of complexities has the negative effect of giving them the impression that programming is easy enough - it could be learned simply by manipulating controls on graphical forms that appear to be concrete. However, when students learn that any event invoked on these virtual controls involve programming event-handling procedures with a logical sequence of code in order for their graphical interfaces to function, they tend to switch off.

We tracked the progress of several classes, of up to 20 students per class, for several years as to their performance in advanced, object-oriented, visual programming. Our test group consisted of students who had taken a class in console-based, procedural programming before their advanced programming class while our control group consisted of students who took an introductory object-oriented visual programming class.

In order to investigate this correlation of programming types and cognitive levels, Azzedine tested 203 students from the 6th grade to college level with the Langeot Test of Cognitive Development, which mirrors Piaget's cognitive development theory, and compared their testing results, and cognitive level, with their results on programming tests. His results, that procedural programming involves a higher cognitive level, relates to our findings (Azzedine, 1987).

Using statistical analysis of average and standard deviation, the average grade of the test group was compared with the average grade of the control group. Our data demonstrates that our test group had a statistically significantly higher grade average (18% higher) than those of our control group. The standard deviance was 14 for the test group and 12 for the control group. Because the standard deviation between the test and control groups was very similar, it demonstrates that our conclusion is well-founded.

Data analysis shows a significant higher performance by the test group and hence supports the hypothesis.

CONCLUSION

The performance and measurement of our test and control groups demonstrate that beginning students, whose introductory programming class is console-based procedural programming, do statistical significantly better in an advanced visual programming class than other beginning students whose introductory programming class is an introductory visual programming class. This significant improvement confirms our hypothesis. Piaget's cognitive development levels, and subsequent studies related to them, that a higher cognitive level is required to grasp the procedural concepts of programming. The use of a higher cognitive level enables students to fully grasp basic programming concepts and to be able to apply them abstractly to a more concrete visual programming paradigm. If students do not rise up to this higher level of cognitive development (by being introduced to and remaining with only visual programming), they tend to remain at the lower concrete cognitive level. Hence, they are unable to think abstractly and they have difficulty applying abstract programming concepts to the coding implementation ("code-behind") of a visual programming application.

AUTHOR INFORMATION

Richard Millham is an assistant professor of computer science at the University of Bahamas. After working in industry for 15 years, he joined academe and has taught for many years. He has worked revising curriculum for the University of Bahamas, Catholic University of Ghana, and the Catholic University of the Sudan. His teaching interests are in the area of software engineering and evolution, service oriented computing, and programming. He holds a BA(Hons) in Computer Science from the University of Saskatchewan, MSc from the University of Abertay in Dundee, Scotland, and a PhD from De Montfort University in Leicester, England. E-mail: richardmillham@hotmail.com. Corresponding author.

Chaker Eid is an assistant professor of computer science at the University of Bahamas. He has over ten years of IT industry experience supplemented with eleven years of IT/mathematics teaching experience. He holds a BSc in Mathematics from the University of Windsor in Canada and two MScs, one in mathematics and the other in computer science, from the University of Detroit-Mercy in the US. E-mail: chakereid@yahoo.com.

REFERENCES

1. Almstrum, V. L. (1994), "Limitations in the Understanding of mathematical Logic by Novice Computer Science Students." Unpublished Dissertation, University of Texas, Austin, Tx.
2. Azzedine, A. (1987), "The relationship of cognitive development, cognitive style and experience to performance on selected computer programming tasks: An exploration." Dissertation Abstracts, B48(6), 1799.
3. Barker, R. J. and E. A. Unger (1983), "A Predictor for Success in an Introductory Programming Class based upon Abstract Reasoning Development." Proceedings of the 14th SIGCSE Technical Symposium on Computer Science Education of the ACM, Orlando, Florida.
4. Becker, H. J. (1982), "Microcomputers in the Classroom -- Dreams or realities?" ERIC(ED217872).
5. Biehler, R. F. and J. Snowman (1986), *Psychology Applied to Teaching*. Houghton Miffin Company, Boston.
6. Cafolla, R. (1987), "The Relationship of Piagetian formal Operations and other cognitive factors to computer programming ability (Development)." Dissertations Abstracts, A47(7), 2506.
7. Chiapetta, E. (1976), "A review of Piagetian studies relevant to science instruction at the secondary and college level." *Science Education*, 60, 253-261.
8. Epstein, H. (1980), "Some biological bases of cognitive development." *Bulletin of the Orton Society*, 30, 46-52.
9. Fletcher, S. H. (1984), "Cognitive Abilities and Computer Programming." EDRS(ED259700).
10. Folk, M. J. (1973), "Influences of Developmental Level on a Child's Ability to Learn Concepts of Computer Programming." Dissertation Abstracts International, 34(3), 1125a.
11. Griffiths, D. H. (1973), "The Study of the Cognitive Development of Science Students in Introductory Level Courses." ERIC(ED096108].
12. Hudak, M. A. and D. E. Anderson (1990), "Formal Operations and Learning Style Predict Success in Statistics and Computer Science Courses." *Teaching of Psychology*, 17(4) 231-234.
13. Little, L. F. (1984), "The Influence of Structured Programming, Gender, Cognitive Development and Engagement on the Computer Programming Achievement and Logical Thinking Skills of Secondary Students." Dissertation Abstracts, A45(6), 1708.
14. Losh, C. L. (1984), "The relationship of student hemisphericity to performance in computer programming courses." Dissertation Abstracts A44(7), 2127.
15. Louden, K. C. (1993), *Programming Languages, Principles and Practice*. PWS Publishing Company, Boston.
16. Monfort, M. and S. A. Martin and W. Frederickson 1990, "Information-processing differences and laterality of students from different colleges and disciplines." *Perceptual & Motor Skills*, 70(1), 163-172.
17. Piaget, J. (1972). Intellectual evolution from adolescence to adult. *Human Development*, 15, 1-12.
18. Pallrand, G.J. (1979), "The transition to formal thought." *Journal of Research in Science Teaching*, 16, 445-451.

19. Pendergast, M.O. "Teaching Introductory Programming to IS Students: Java Problems and Pitfalls", *Journal of Information Technology Education*, vol 5, (2006)
20. Petrushka, D. (1984), "A Study of the effect of content on the ability to do syllogistic reasoning: An investigation of transferability and the effect of practice." Unpublished doctoral dissertation, Rutgers University, NJ.
21. Renner, J. and R. Grand and J. Sutherland (1978), "Content and concrete thought." *Science Education*, 62, 215-221.
22. Schwebel, M. (1975), "Formal operations in first year college students." *Journal of Psychology*, 91, 133-141.
23. Sund, R. B. (1976), "Piaget for educators: a multimedia program." In Brooks, R 1978 "The relationship between Piagetian Cognitive" *Journal of Information Systems Education*, Vol 13(1) 6 6
24. White, G. L. (2002) "A Theory of the Relationships between Cognitive Requirements of Computer Programming Languages and Programmers' Cognitive Characteristics" *Journal of Information Systems Education*, Vol 13(1), pp 60-66
25. Zak, D. (2005) *Programming with Microsoft Visual Basic 2005*, Cengage, Boston.

