ED 398 892 IR 018 072

AUTHOR	Myers, J. Paul, Jr.; Munsinger, Brita
TITLE	Cognitive Style and Achievement in Imperative and Functional Programming Language Courses.
PUB DATE	96
NOTE	10p.; In: Call of the North, NECC '96. Proceedings of the Annual National Educational Computing Conference (17th, Minneapolis, Minnesota, June 11-13, 1996), see IR 018 057.
PUB TYPE	Reports - Research/Technical (143) -- Speeches/Conference Papers (150)
EDRS PRICE	MF01/PC01 Plus Postage.
DESCRIPTORS	*Achievement Gains; *Cognitive Style; Computer Science Education; *Correlation; Higher Education; *Programming; Programming Languages; Scores; Skill Development; Undergraduate Students
IDENTIFIERS	*Learning Style Inventory (Kolb); Paradigmatic Responses; Trinity University TX

ABSTRACT

This paper investigates the relationship between learning style and programming achievement in two paradigms: imperative and functional. An imperative language achieves its effect by changing the value of variables by means of assignment statements while functional languages rely on evaluation of expressions rather than side-effects. Learning style was measured by the Kolb Learning Style Inventory, and achievement was measured by programming course grades. Subjects were 32 undergraduate Computer Science students at Trinity University (Texas). General ability in programming proved much stronger an effect than learning style on success in programming courses. Success in one course highly correlated with success in the other, regardless of the language. Though difficult to discern from this larger effect, small effects from learning style were enough to suggest directions for further research. (Contains 23 references.) (AEF)

Paper

Cognitive Style and Achievement in Imperative and Functional Programming Language Courses

J. Paul Myers, Jr.
Department of Computer Science
Trinity University
210.736.7398
pmyers@cs.trinity.edu

Brita Munsinger
Department of Computer Sciences
University of Texas
bmunsing@cs.utexas.edu

Key Words: **learning style, programming paradigms, Kolb Learning Style Inventory, achievement, behaviorism, Scheme, C**

Abstract

We investigated the relationship between learning style and programming achievement in two paradigms: imperative (C) and functional (Scheme). Learning style was measured by the Kolb LSI; achievement was measured by programming course grades. Success in one course highly correlated with success in the other, regardless of language used. Though difficult to discern from this larger effect, small effects from learning style were enough to suggest further research.

Background

> The development of programming languages and methods, and the teaching of them, have up to now hardly been linked to a psychological study of the activity of programming, and this can account for certain failures...psychology must go beyond the procedural aspect of programming; it must take into account those other styles which, even if they are not new, are becoming more and more important nowadays due to the variety of applications and the training that programmers receive (Hoc et al., 1990).

Teaching programming is a challenge given the variety of languages used and the controversies surrounding their appropriateness for education. Study of the language-learning process is necessary to understand how the process can be improved. In this study, the possible connection between learning style and achievement in two programming styles is explored.

Theories of Learning Styles

Much educational and psychological research has been devoted to learning styles. Some troubles in teaching have been attributed to a mismatch between the structure of schooling and the peculiarities of how an individual learns. Studies have shown that students with particular learning styles perform better in different environments (Dunn et al., 1989). Wu (1993) states, "Individual styles ... not only result in preferences for different modes of presentation of learning materials and of analogies, but also lead to individual differences in the organization of semantic knowledge."

Learning style is a nebulous concept; definitions range from the simple, "how a student learns and likes to learn" (Keefe, 1988), to the precise notion of a...

> ... composite of characteristic cognitive, affective, and physiological factors that serve as relatively stable indicators of how a learner perceives, interacts with, and responds to the learning environment. It is demonstrated in that pattern of behavior and performance by which an individual approaches educational experiences. Its basis lies in the structure of neural organization and personality (Keefe, 1988).

Early learning style theories, based on behaviorism, shifted from the simple stimulus/response model to internal cognitive factors. Jung influenced the theory, classifying people into four types: thinkers, feelers, sensors, and intuitors (Bokoros et al., 1992). Later work in the 1970s, by Gregorc, identified four learning characteristics: concrete, abstract, sequential, and random. He paired these, ending up with four learning style types (AASA, 1991).

Kolb developed a model of learning also based on four stages: concrete experience, reflective observation, abstract conceptualization, and active experimentation. Within each stage people learn respectively by feeling, observing, thinking, and doing. Scores on these scales can be used to find a single type describing a person's learning style: either accommodator, diverger, converger, and assimilator (Kolb, 1985).

Each type tends toward a certain kind of career, Kolb asserts. Convergers are "best at finding practical uses for ideas and theories." Divergers see situations from many points of view, and are good at idea-generation and imagination. Assimilators are logical, working well with abstract ideas. Accommodators prefer working with people and learning from experiences. These learning styles have implications for tailoring education to the needs of the student (Almstrum and Wu 1993).

Computer Programming Paradigms

An *imperative language*, including C, Pascal, and FORTRAN, "achieves its effect by changing the value of variables by means of assignment statements" (Wilson & Clark, 1993). These commands update variables held in storage (the Latin "imperare" means "to command"), a process known as "side-effect." Imperative languages use sequence, selection, and iteration, making them very efficient for execution of programs on von Neumann architecture. This is true especially of C, a high-level language, designed to behave like a low-level language for flexibility and efficiency (Wilson & Clark, 1993).

Functional languages, such as LISP and its derivative Scheme, rely on evaluation of expressions rather than side-effects. The input-to-output conversion simply uses expressions, functions, and declarations without variables; though some functional languages (e.g., Scheme) rely on side effects for certain functions. Such programs are controlled recursively rather than iteratively.

While there are other paradigms (e.g., logic and object-oriented), imperative and functional languages (respectively, C and Scheme) are the focus of this study.

Learning Styles and Computer Programming

Programming aptitude has been measured (e.g., an IBM test focused on general verbal and spatial abilities and mathematical reasoning); but it is unclear how well this predicts performance in programming tasks (Palormo, 1964; Tromp, 1989). There are relations between mathematical (e.g., measured by the SAT) and programming ability, but they largely disappear after factoring out general ability (Tromp, 1989). As recently as the 1970s, aptitude for programming jobs was assessed by these tests (Luftig, 1975); nowadays, however, an understanding of how people learn different computational models is necessary to accommodate the present diversity of languages.

There is a need for investigating how the ways people learn, not just aptitude, may influence productivity in various programming languages (Hoc et al., 1990).

> There are many ways to do research on programmers, but the discipline of controlled psychologically-oriented experiments produces reliable and authoritative results....A major research project to investigate the impact of programming language design on productivity and error rates and to develop an understanding of the cognitive processes in programming could have a dramatic influence on future programming languages (Soloway & Iyengar, 1986).

Indeed, once we better understand how people learn to program and fare in different programming environments, new languages could be designed to make better use of human abilities.

Mayer has suggested that difficulties some students have is that learning programming involves a great deal of precision-intensive problem solving. In some cases, the students have what he calls a "fragile knowledge of domain": students have trouble applying their own knowledge effectively to the problem at hand. There are also problems with confidence in abilities and feelings of control or mastery; and he has found that a structured approach works better than letting students discover concepts on their own (Mayer, 1988).

Van Merrienboer explored more directly the relationship between learning style and success in programming with two studies on the specific reflection-impulsivity cognitive style dimension. Reflectives tend to spend more time examining problems, while impulsives prefer to jump into the middle and are less systematic in gathering information. He found that the reflectives performed better at comprehension of programs; and so he investigated ways to get impulsives to move to a more reflective strategy. Having impulsives do coding assignments that require the generation of new programs is more effective, while reflectives benefit from completing partial programs (van Merrienboer, 1988, 1990).

Kagan found a link between personality and achievement in programming: "Significant correlations obtained between personality scales and achievement were computed *after* variance attributable to inherent aptitude was removed" (Kagan, 1988). She also found "among both males and females, achievement in FORTRAN was significantly related to a tendency to organize information analytically"; and she noticed a tendency of successful students to be "compulsive," "perfectionist," and "attentive to detail" (Kagan, 1988).

Two of Piaget's stages, concrete operational and formal operational, were the subject of a study conducted by Hudak and Anderson. They investigated whether level of cognitive development and learning style, as measured by the Kolb, were related to success in Statistics and Computer Science (CS) courses. Hypotheses were confirmed that people with more cognitive development, having reached the formal operational stage, would have the tools needed to understand those concepts and that more abstract learning styles helped with achievement (Hudak & Anderson, 1990).

Foreman found correlations between program understanding and the cognitive characteristics of field independence, reasoning, and direction following: "alternate instructional approaches would help different cognitive styles and aptitudes, to learn programming skills more quickly and effectively" (Foreman, 1990).

In other studies, word problem and algebra skills aided in computer programming but group learning of programming did not differ from individual learning (Shute, 1991; Webb et al., 1986).

The present study is perhaps closest in spirit to Wu's in which cognitive learning style, as described by Kolb, was correlated with novice learning of recursion. That study demonstrated that "novice programmers with abstract learning styles performed better than those with concrete learning styles when learning recursion" (Wu, 1993); the concrete conceptual model was likened to a "glass box," the abstract conceptual model to a "black box," with respect to awareness of the underlying machine.

Thus, in the present study, certain results could be anticipated. Since programming in the functional language Scheme is replete with recursion (and black-box functionality), one would expect success to be associated with an abstract learning style. And the closeness of the procedural C language to the von Neumann machine ("glass box") would suggest correlation with a concrete learning style.

Procedure

Subjects

Subjects were undergraduate CS students at Trinity University, a small private liberal arts institution in Texas. Most were sophomores or juniors; most had taken introductory CS courses at Trinity. Thirty-two subjects were selected from the courses in Functional Language Programming, Assembly Language Programming, and Formal Languages, giving a range both in age and background. All students were assumed to have taken CS 311–312 (Problem Solving and Algorithm Design I, II) and CS 316 (Functional Language Programming).

Method

Each subject received a packet containing a consent form, a sheet for grades, and a copy of the Kolb LSI for self-administration. The grade sheet was to be used for confidentially recording grades in three courses: CS 311–312 (C) and CS 316 (Scheme). The advisor for this project recorded grades and removed the consent forms before data were transferred to the student researcher; thus the data were anonymous.

Means and standard deviations of grades were calculated for each course, and grades for each student were standardized into a 4.0 scale (by subtracting the class mean from each grade and dividing by the standard deviation) to facilitate comparison across classes. Scores from the Kolb LSI were translated to percentiles. The resulting standardized numbers were used as data points in the statistical analysis. Course grades were collected for three classes, CS 311, CS 312, and CS 316, but only grades in CS 312 and CS 316 were used in the analysis. CS 311 grades were deleted for three reasons:

1. Students in CS 312 and 316 would have prior experience in programming, CS 311 students might not.
2. Some students in CS 311 had been taught with Pascal, others with C; they were not distinguished in the data collection.
3. Many students had not taken CS 311 (e.g., by AP credit).

Statistical Analysis

The data on grades and learning style were analyzed using a linear regression model, chosen over more sophisticated models due to the small sample size. The data were plotted in all combinations of the CS courses with each of the four learning styles, giving twelve plots, each only mildly linear. The grades for the second introductory CS class were plotted against those of the functional language course, giving a more distinct linear pattern.

C statistics were calculated for all possible combinations of independent variables in two separate models (SAS notation) where learning style scores are RO (reflective observation), CE (concrete experience), AC (abstract conceptualization), and AE (active experimentation):

1. CS312 = RO CE AC AE CS316
2. CS316 = RO CE AC AE CS312.

The C statistic, a function of the error in the model, provides a way to compare different models in terms of their explanatory value, taking into consideration how many variables are in the model. More variables on the right-hand side may help the model explain more variance in the left-hand variable, but some do little to help and should be dropped. In general, the simpler the model, the better (Bowerman and O'Connell, 1990).

For the first model, the combination of independent variables with the optimal combination of R^2 and the C statistic was RO and CS 316. In the second model, the best combination was CE and CS 312. Stepwise regressions yielded similar results, although in model 1, RO did not meet the entry significance level. Stepwise regression involves performing several regression models in succession with different sets of independent variables; variables are added if they add significantly to the model and variables are removed that do not meet the significance level. The final model from a stepwise regression is the "best" combination of independent variables, based on how much they contribute to the overall model (Bowerman and O'Connell, 1990).

Regular linear regressions were run on the independent variables suggested by the above tests. In both cases the overall models were statistically significant, with p-values below .04. The p-values for t-tests for individual coefficients were higher, though; and only the grades from CS 312 and CS 316 were significant at the .05 level. P-values measure the probability that the statistics indicate that a variable or model is significant, when it in fact is not significant. Lower p-values indicate increased confidence in the model.

Models were run including all 5 independent variables plus interaction terms: the 6 possible combinations of pairs of learning style dimensions (RO, CE, AC, AE). The overall significance of the models dropped considerably; in fact, both models had p-values for the overall F-test above .5. Partial F-tests were done for both to see if the interactions and three additional learning style dimensions should be dropped. In all four cases, the answer was a resounding yes, with p-values above .8.

More tests were run with the earlier models that included two independent variables. Residual plots showed appropriately distributions, although in one case (RESID1*RO) the variance was not constant. Residual plots show the error values (differences between actual and predicted values for the independent variable) for each model. There should be no discernible pattern in the errors, since otherwise, that would mean there were more that is not explained by our model. The stem-and-leaf plot, box plot, and normal probability plot do not indicate any violations of the regression assumptions for model 1. However, the stem-and-leaf plot for model 2 is top-heavy. The box plots appear reasonable, though. The above three plots are used to check whether the data are normally distributed, which is one of the predictions of the regression model (Bowerman and O'Connell, 1990).

Based upon these analyses, the best two models appear to be (in SAS notation):

1. CS312 = RO CS316
2. CS316 = CE CS312.

Within this model, the results indicate that grades in one course are the best single predictor of performance in the other. However, there may be an additional effect from learning style. Though it is not nearly as strong, scores on two of the dimensions of the Kolb LSI predict some of the variance in grades in CS 312 and CS 316.

Conclusions

General ability in programming proved much stronger an effect than learning style on success in programming courses, regardless of the language. Yet learning style did contribute some predictive value to the models. Also, different learning styles were more significant predictors in one language than another; high scores on the RO (reflective-observation) scale of the Kolb were associated with higher grades in CS 312. The RO scale measures a person's tendency to patiently observe and act carefully; this might be associated with success in C since the syntax of C programs is complex and coding requires careful work (and patience).

The CE (concrete-experience) scale was associated with skill in CS 316. People with high scores on CE tend to be open-minded and adaptable to change. Since programming in Scheme is so different from the procedural languages many students first encounter, that openness may help with readily accepting a new paradigm. In addition, functional languages are also called applicative languages, since functions are directly applied to data. This directness may help to illuminate the structure particularly well for someone who has high scores on the CE scale of the Kolb.

We see here, then, results that differ from those that were earlier anticipated along the lines of Wu's study (1993). Here success in the procedural language C was associated with a RO cognitive style, not with a concrete style. Indeed, in the present study, a CE cognitive style was correlated with success in the functional language! Perhaps the comprehensive rigors of learning an entire language are not at all comparable to learning a specific aspect of that (or any) language—e.g., recursion, as in Wu's study. The above speculative comments may serve to illuminate these points; but certainly we can see the desirability of further research into these relationships.

That a learning style effect is difficult to separate from general ability should not leave us discouraged. The fact that such an effect exists should motivate more research into this question, since it provides a possible way to make CS education more effective for novices.

As this is a relatively unexplored area, there is ample room for more studies. Acknowledging limitations in the present study and suggesting alternate strategies should prove useful to those planning further work in this arena. One limitation of this study was the lack of a large student pool. A larger-scale study would be necessary before many conclusions could be drawn. Or, especially given a small N, it would be desirable to determine the subjects' subjective assessment or preference for programming language in addition to the objective measure of grades. And programming paradigms other than imperative and functional should be considered in future studies.

An orientation toward expert subjects rather than the novices here might be especially fruitful: bypass students altogether and test faculty. Ideally students would be studied who, with no prior programming experience, would see either an imperative or a functional language *as their first experience*. Such are increasingly difficult to find as many college students have had prior programming (usually with an imperative language). Many faculty, on the other hand, who teach programming classes have strong preferences for which language is "best" as a first one for novices. Testing learning styles and relating them to *preferred* languages might yield interesting results. Presumably the bias inherent in their own first exposure would be eliminated or reduced by years of experience and consideration given to this issue, especially if the subjects had taught both styles. Then, of course, the study would not be about which is more suitable for students based on students' cognitive style, but rather, what do experts tend to recommend based on their own cognitive style? And ... do the same cognitive styles influence novices and experts in the same ways?

Bibliography

Almstrum, Vicki L. and Cheng-Chih Wu. "The CSedRes Toolbox: A Resource to Support Research in Computer Science Education." *SIGCSE Bulletin*, 25, No. 4 (1993), 21–26.

AASA (American Association of School Administrators), *Learning Styles: Putting Research and Common Sense into Practice*, Arlington, VA, 1991.

Bokoros, Michael A., Marc B. Goldstein, and Mary M. Sweeney. "Common Factors in Five Measures of Cognitive Style." *Current Psychology: Research and Reviews*, 11, No. 2 (1992), 99–109.

Bowerman, Bruce L. and Richard T. O'Connell. *Linear Statistical Models: An Applied Approach*. Boston: PWS-KENT Publishing Company, 1990.

Dunn, Rita, Mary Cecilia Giannitti, John B. Murray, Ino Rossi, Gene Geisert, and Peter Quinn. "Grouping Students for Instruction: Effects of Learning Style on Achievement and Attitudes." *The Journal of Social Psychology*, 130, No. 4 (1989), 485–494.

Foreman, Kim H. "Cognitive Characteristics and Initial Acquisition of Computer Programming Competence." *School of Educational Review*, 2 (1990), 55–61.

Hoc, J.-M, T.R.G. Green, R. Samurçay, and D.J. Gilmore, eds. *Psychology of Programming*. London: Academic Press, 1990.

Hudak, Mary A. and David E. Anderson. "Formal Operations and Learning Style Predict Success in Statistics and Computer Science Courses." *Teaching of Psychology*, 17, No. 4 (1990), 231–234.

Kagan, Dona M. "Learning How to Program or Use Computers: A Review of Six Applied Studies." *Educational Technology*, 28, No. 3 (1988), 49–51.

Keefe, James W., ed. *Profiling and Using Learning Style*. Reston, VA: National Association of Secondary School Principals, 1988.

Kolb, David. *Learning Style Inventory*. Boston: McBer and Company, 1985.

Luftig, Milton. *Computer Programmer: The Complete ARCO Test-Tutor*. New York: ARCO, 1975.

Mayer, Richard E., ed. *Teaching and Learning Computer Programming: Multiple Research Perspectives*. Hillsdale, NJ: Lawrence Erlbaum Associates, Publishers, 1988.

Munsinger, Brita. *Learning Style and Achievement in Imperative and Functional Programming Language Courses*. San Antonio, Texas: Trinity University Senior Thesis, 1995.

Palormo, Jean Maier. *Computer Programmer Aptitude Battery*. Chicago, IL: Science Research Associates, Inc., 1964.

Shute, Valerie J. "Who is Likely to Acquire Programming Skills?" *Journal of Educational Computing Research*, 7, No. 1 (1991), 1–24.

Soloway, Elliot and Sitharama Iyengar, eds. *Empirical Studies of Programmers*. Norwood, NJ: Ablex Publishing Corporation, 1986.

Tromp, Th. J. M. *The Acquisition of Expertise in Computer Programming*. Amsterdam: Uitgeverij Thesis, 1989.

van Merrienboer, Jeroen J. G. "Instructional Strategies for Teaching Computer Programming: Interactions with the Cognitive Style Reflection-Impulsivity." *Journal of Research on Computing in Education*, 23, No. 1 (1990), 45–53.

van Merrienboer, Jeroen J. G. "Relationship Between Cognitive Learning Style and Achievement in an Introductory Computer Programming Course." *Journal of Research on Computing in Education, 21*, No. 2 (1988), 181–186.

Webb, Noreen M., Philip Ender, and Scott Lewis. "Problem Solving Strategies and Group Processes in Small Groups Learning Computer Programming." *American Educational Research Journal*, 23, No. 2 (1986), 243–261.

Wilson, Leslie B. and Robert G. Clark. *Comparative Programming Languages*. Wokingham, England: Addison-Wesley Publishing Company, 1993.

Wu, Cheng-Chih. *Conceptual Models and Individual Cognitive Learning Styles in Teaching Recursion to Novices*. Austin, TX: University of Texas at Austin Doctoral Dissertation, 1993.