Communications of the Association for Information Systems

Volume 5 Article 2

1-2001

DO SEQUENCE AND CONCURRENCY MATTER?: An Investigation of Order and Timing Effects on Student Learning of Programming Languages

Andrew Urbaczewski
Washington State University, andrewu@wsu.edu

Bradley C. Wheeler
Indiana University

Communications of the Association for Information Systems

Volume 5, Article 2
January 2001

DO SEQUENCE AND CONCURRENCY MATTER?: An Investigation of Order and Timing Effects on Student Learning of Programming Languages

Andrew Urbaczewski
School of Accounting, Information Systems, and Business Law
Washington State University
andrewu@wsu.edu

Bradley C. Wheeler
Kelley School of Business
Indiana University

EDUCATION

DO SEQUENCE AND CONCURRENCY MATTER?: An Investigation of Order and Timing Effects on Student Learning of Programming Languages

Andrew Urbaczewski
School of Accounting, Information Systems, and Business Law
Washington State University
andrewu@wsu.edu

Bradley C. Wheeler
Kelley School of Business
Indiana University

ABSTRACT

IS educators often struggle with curriculum issues including timeliness and completeness of the curriculum. While model curricula suggest that programming courses should be a part of an IS undergraduate degree, little guidance is offered as to the order and timing of these courses. A longitudinal survey of students in programming courses was used to assess whether sequence or concurrency explained any variance in perceptual performance measures. Sequence of programming courses did not hinder student performance, and concurrency actually improved performance for Visual Basic. Insights from the study provide guidance for curricular design issues regarding the sequencing and timing of programming courses.

Keywords: Programming, IS Education

I. INTRODUCTION

programming. v. *"The designing, scheduling, or planning of a program." [Webster, 1993]*
"A pastime similar to banging one's head into a wall, but with fewer opportunities for reward." [FOLDOC, 2000]

The curriculum for undergraduate Information Systems (IS) education is constantly evolving to keep pace with new technologies. Educators are engaged in a seemingly persistent state of curriculum redesign to ensure that students gain the state-of-the-art technological skills required of IS professionals. Programming languages represent a foundational part of that curricular evolution. It is common today for recruiters to seek students who have skills in multiple programming languages.

In accordance with recent curriculum design guidelines, such as IS '95 [Couger et al. 1995] and IS '97 [Davis et al. 1997], and the forthcoming IS 2000, new technologies are to be incorporated into university curricula for keeping the content as contemporary as possible. In the rush to design curricula that maximize exposure to new technologies, it is possible that the factors which maximize learning efficacy may have been overlooked.

Do sequence and concurrency matter in developing programming skills? Prior research on *sequence* found mixed results for student performance [Manns and Carlson 1992, Rosson and Alpert 1990]. Some advocate learning an object-oriented programming language (OOPL) first, while others contend that a third generation language (3GL) should come first [Powell 1997]. Veteran programmers learned programming concepts in the older, procedural second and third generation languages before learning 4GLs or visual programming environments. Students now can learn a 4GL or OOPL before learning or without ever learning a 3GL. Moreover, with strong demand for the undergraduate IS major, it is often difficult for students to arrange schedules with any degree of pedagogical optimality. Rather, they take the courses they can get and fit them into their schedule. Students sometimes take two or three programming languages concurrently during the same semester. Does this lack of enforced language sequence affect students' ability to acquire programming skills?

The second issue, *concurrency*, or learning two or more programming languages simultaneously, could either hinder or could possibly help in acquiring programming skills. Arguments for hindrance advocate that cognitive overload would diminish students' abilities to grasp the differing programming syntax, functions, and techniques of multiple languages. The challenge would be similar to learning two spoken foreign languages, like an American student learning French and German, simultaneously. Alternatively, learning two programming languages concurrently could be complimentary in grasping the higher level programming concepts, such as loop, branch, and sequence, even though different programming languages express these ideas through different syntax.

This study explored two primary research questions:

1. Do students learn languages better when they are taught in a particular order, such as, 4GLs before 3GLs or vice versa?
2. Do students learn programming languages better when they are taken sequentially rather than simultaneously?

II. FOUR PERSPECTIVES ON LANGUAGE SEQUENCE

While the IS ‘97 curriculum guidelines help educators decide *which* programming courses to teach, they provide no guidance for the optimal sequence in which they should be taught. The guidelines state that “graphic programming environments should be explored” and that “program design methods and strategies including top-down implementation will be discussed and implemented” [Davis et al. 1997]. The baseline case could argue that the sequence in which the languages are learned is irrelevant for programming skill development. We offer four alternative perspectives regarding possible curriculum sequences:

1. Evolutionary sequence
2. Difficulty Sequence
3. As needed sequence
4. Filtering sequence

EVOLUTIONARY SEQUENCE

As computer programming was brought to the masses, the languages evolved in generations. First generation languages (1GLs) like machine language and second generation languages (2GLs) like assembler were used by early programmers, but their complexity and difficulty in debugging left programmers needing better tools for software development. 3GLs and 4GLs came later, as did OOPLs and visual programming languages. In a business school environment, seldom is machine language or assembler part of the curriculum. They tend to fall in the domain of Computer Science programs. The oldest 3GL generally encountered in business schools is COBOL, and then perhaps C. These languages are both top-down implementation environments. Because IS '97 also requires instruction in graphical programming environments, students should also be exposed to another language. This language is often Visual Basic (VB) or sometimes Powerbuilder™. Since this sequence is how the languages evolved, perhaps it makes the most sense for students to acquire 3GLs prior to 4GLs prior to visual languages.

DIFFICULTY SEQUENCE

A second school of thought is that students should take the classes in a manner that eases them into the world of programming. They would begin with the most English-like language and then progress to more cryptic languages. This approach involves taking the languages in descending order of their resemblance to natural language. COBOL is the most English-like of the widely-used programming languages, followed by VB. C, however, is the most cryptic of the major languages and would be taken last.

AS NEEDED SEQUENCE

In business today, many legacy programs tend to be written in COBOL. While some organizations used Y2K preparations as an opportunity to recode these applications into other languages, many business systems are still written in COBOL. Newer programs, especially those written for client-server data access, are often written in a visual language like Visual Basic. Visual Basic is often chosen because of the speed with which programs can be written. For more intensive programming efforts, C or C++ is often required. Assuming that students going out into the job market will be working on newer systems first and maintaining legacy systems second (even more so since the recent maintenance effort with Y2K), they are more likely to need VB skills first, then C, and perhaps COBOL later.

FILTERING SEQUENCE

Finally, a fourth sequence could be designed to serve objectives other than maximizing student learning. Programming course sequence could be used to create a significant hurdle early in an IS degree program as a weed-out mechanism for dissuading students who may not have an aptitude for acquiring IS technical skills. This model would require students to learn the most difficult programming languages first, followed by the more English-like languages.

The four perspectives are summarized in Table 1.

Table 1. Alternate Sequencing of Programming Languages

Perspective	**Sequence**
Evolutionary	COBOL, C → C++, VB → PowerBuilder
Difficulty	PowerBuilder, VB → COBOL → C →C++
As Needed	VB, PowerBuilder → C++, C →COBOL
Filtering	C → C++ → COBOL → VB

III. HYPOTHESES

Veteran programming course instructors frequently hear student concerns regarding their difficulties in learning a particular language. One concern comes from students who have taken no prior programming courses. They sometimes perceive that they are less prepared for the course than other students who already learned one or more programming languages. They worry that their objective performance in the class will suffer because they are being compared to students with prior programming experience. The second concern is from students who are familiar with another computer language and are having difficulty with the current course. Often these students are attempting to learn two languages at the same time.

The following hypotheses are drawn from the four perspectives on sequence and the concurrency concerns expressed by students. They are asserted for students who are completing a particular programming language course:

H1: Students who took any prior programming course demonstrate greater performance with the language than those who have not taken any prior programming course (Sequence).

H2: Students who are taking only one programming course demonstrate higher performance with the language than those who are taking multiple, concurrent programming courses (Concurrency).

In addition to course performance, students also develop a perceived level of comfort with using a programming language. While this perception does not equate to actual skill demonstration, it does provide an additional measure of perceived learning or confidence in applying the course material. Therefore, we propose *Perceived Comfort* as a dependent measure for student mastery of programming material.

H3: Students who took any prior programming language course report greater comfort with the language than those who did not take a prior programming course (Sequence).

H4: Students who take only one programming course will report greater comfort with the language than those who are taking multiple, concurrent programming courses (Concurrency).

IV. METHODOLOGY

Students were drawn from nine introductory programming courses over three semesters at a large Midwestern university. Three hundred and forty-one students responded to a survey at the end of the semester in which they were completing the programming courses. These courses were:

1. *Introduction to Visual Programming*,
2. *Introduction to COBOL Programming*, and
3. *Introduction to C Programming*.

A particular student may have been in one, two, or three of the courses depending upon her schedule. Students responded anonymously to remove social desirability bias as a threat to internal validity [Campbell and Stanley 1963]. The survey was taken without compensation to the students, because it simply required a few minutes of their time at the beginning of a class period. One response was determined to be incomplete and unusable. Subjects were asked to indicate at the top of the survey if they had completed the survey in another course. Nineteen students indicated multiple questionnaires, and their surveys were eliminated to avoid double counting.

Subjects reported not only on the course in which they were currently enrolled but also on all other programming courses they had taken for a grade at the collegiate level or above. These prior and concurrent programming courses were coded as 0=no and 1=yes to serve as the independent variables for hypothesis testing.

Two dependent variables were measured. Self-reported *Course Grade* was used as a surrogate for actual course performance. It was measured as the self-reported letter grade for the student from prior courses and the expected grade for the current course.[1] This letter grade was then converted into numerical equivalents for data analysis. The second dependent variable, *Perceived Comfort* level*,* measured students' perceived skill with the language. Students may earn a high grade in a course with relatively low mastery of the material, and vice versa. This effect is often due to additional factors which figure into the final grade but do not directly measure skill mastery, such as attendance and class participation.

The survey also captured other demographic data that could likely affect the dependent measures. These included *Attendance* in class, cumulative *Grade Point Average* (*GPA*), and *Prior Experience* with the programming language (e.g., internship, hobby, etc).

V. ANALYSIS AND RESULTS

Each programming course, e.g., *Introduction to Visual Basic*, was analyzed separately pooling all responses across three semesters for the course. Course Grade and Perceived Comfort were significantly correlated ($n=611, r=.439, p<.001$), therefore, statistical analyses were conducted via MANCOVA. The three demographic variables were specified as covariates in a separate MANCOVA model for each dependent variable with prior programming course(s) and concurrent programming course(s) serving as the two-level factors.

In all the MANCOVA models, the omnibus multivariate scores for the three covariates were all significant at the .05 level. There were no significant interaction effects between the independent variables at the .05 level. Results from the individual F tests are shown in Table 2.

1 The actual course grade for prior courses was known by the students while the final course grade for the current programming class was based on their expected grade. Given that most syllabus items except for the final had been graded by the time of the survey, the students were assumed to base their perception on performance during the semester prior to the final exam.

For the Visual Basic course, all three demographic variables were significant at the .05 level in explaining Course Grade (GPA *F*(1,226)=15.549, *p*=.000; Attendance *F*(1,226)=20.414, *p*=.000; Prior Experience *F*(1,226)=6.017, *p*=.015; R^2=.197). Neither factor nor the interaction term were significant in explaining course grade.

For the Perceived Comfort variable, Prior Experience was the only significant demographic variable (*F*(1,234)=5.689, *p*=.018). Prior Course and Concurrent Course (*F*(1,234)=5.086, *p*=.025; *F*(1,234)=5.689, *p*=.018; R^2=.108) were both significant though the interaction term was not. Table 2 reports the descriptive statistics for both dependent variables.

Table 2 Descriptive Statistics for the Visual Basic Course

	No Prior Course	**Had a Prior Course**
Course Grade		
No Concurrent Course	n=99 3.0434 (*.6974*)	n=84 3.3107 (*.5720*)
With a Concurrent Course	n=42 3.2524 (*.5071*)	n=8 3.000 (*.5345*)
Perceived Comfort		
No Concurrent Course	n=102 4.5588 (*1.1988*)	n=85 5.1765 (*.9534*)
With a Concurrent Course	n=46 5.1304 (*1.1471*)	n=8 5.6250 (*.9161*)

For the COBOL course, GPA and Attendance were significant at the .05 level in explaining Course Grade (*F*(1,116)=40.532, *p*=.000; *F*(1,116)=4.757, *p*=.031; R^2=.338). Neither factor nor the interaction term was significant in explaining Course Grade. For the Perceived Comfort variable, no demographic or factor was significant. Table 3 reports the descriptive statistics for both dependent variables.

Table 3 Descriptive Statistics for the COBOL Course

n, mean, (st. dev)	**No Prior Course**	**Had a Prior Course**
<u>Course Grade</u>		
No Concurrent Course	n=31	n=75
	3.3516 (*.6501*)	3.2760 (*.6726*)
With a Concurrent Course	n=12	n=5
	2.9167 (*1.1946*)	3.000 (*.7071*)
<u>Perceived Comfort</u>		
No Concurrent Course	n=33	n=76
	4.6970 (*1.4892*)	5.0921 (*1.0976*)
With a Concurrent Course	n=12	n=6
	4.9167 (*.7930*)	5.3333 (*.5164*)

For the C course, all three demographic variables were significant at the .05 level in explaining course grade (GPA *F*(1,241)=55.947, *p*=.000; Attendance *F*(1,241)=6.660, *p*=.010; Prior Experience *F*(1,241)=9.948, *p*=.002; R^2=.261). Neither factor nor the interaction term was significant in explaining course grade.

For the Perceived Comfort variable, Prior Experience was the only significant demographic variable (*F*(1,258)=16.089, *p*=.000; R^2=.08). Prior Course, Concurrent Course, and the interaction term were not significant. Table 4 reports the descriptive statistics for both dependent variables.

Table 4 Descriptive Statistics for the C Course

n, mean, (st. dev)	**No Prior Course**	**Had a Prior Course**
Course Grade		
No Concurrent Course	n=140 3.43.14 (*.6049*)	n=62 3.4710 (*.5756*)
With a Concurrent Course	n=42 3.4095 (*.6559*)	n=4 3.3500 (*.4041*)
Perceived Comfort		
No Concurrent Course	n=144 4.9792 (*1.0740*)	n=62 5.0645 (*1.0381*)
With a Concurrent Course	n=53 5.1132 (*1.3820*)	n=6 5.8333 (*.9832*)

VI. DISCUSSION

LIMITATIONS

Limitations of this research include its reliance on perceptual and self-reported data rather than objective demonstrations of programming skill proficiency. Expected Course Grade was used as a surrogate measure for actual course grade. Since programming course grades are highly reliant on homework and examinations that require demonstration of programming skill, we believe course grade is a reasonable surrogate measure for the objectives of this research. In addition, this research only investigated courses in three programming languages, which did not include Java, PowerBuilder, or other popular development tools. The sample size for COBOL is smaller than for the other two languages since it was not a required course and attracted smaller enrollments. Finally, each instructor has his or her own teaching style. While multiple instructors were involved in teaching the three courses across the three semesters, the research did not measure stylistic differences among the instructors.

DEMOGRAPHIC COVARIATES

As expected, the demographic variables of prior GPA, Attendance in class, and any Prior Experience accounted for most of the variance in explaining Course Grade. Students who historically make an "A" in courses are likely to do so again. Students who attend class regularly are likely to do better than those who do not. A further examination of the demographic variables reveals an interesting pattern. GPA and Attendance consistently explained variance in Course Grade for all three languages, but neither explained significant variance for Perceived Comfort with the language. Prior Experience was the only significant demographic variable for Perceived Comfort. Thus, students' Perceived Comfort with a programming language was not affected by prior GPA or by class attendance.

HYPOTHESES

The research questions and hypotheses address the efficacy of sequence and concurrency beyond the explanations provided by the demographic factors. The interpretation of these data does not provide support for either H1 nor H2. Prior or concurrent programming courses did not significantly contribute to explaining student performance in programming courses as measured by self-reported Course Grade.

The findings for H3 and H4, Perceived Comfort, are mixed. Students who took a prior programming course reported significantly higher Perceived Comfort with the Visual Basic programming language than students without a prior course, thus providing some support for H3. In contradiction of H4, students who were taking a COBOL or C course concurrently with Visual Basic reported significantly *greater* Perceived Comfort than did students who were not taking another concurrent programming course. Neither COBOL nor C provided any support for H3 or H4.

We conjecture from this data that instruction in a graphical programming environment, such as Visual Basic, may benefit from prior or concurrent instruction in a 3GL. Third generation languages, such as COBOL and C, provide a strong introduction to foundational programming concepts (e.g., loop, branch, and sequence) that may be less conceptually obvious in a visual language. While the measure of Course Grade did not find support for this conjecture, the separate and directionally consistent data for Perceived Comfort suggests that exposure to these 3GLs increased students' perceived comfort with the language.

VII. CONCLUSIONS AND FUTURE RESEARCH DIRECTIONS

Our two research questions asked if sequence and concurrency of programming courses matter in learning programming skills. Our answer is no. We conclude from this data that curriculum designers need not be overly concerned in sequencing programming language courses. Similarly, we found no hindrance for concurrent enrollment in different programming courses, and concurrent courses may actual provide a benefit for learning Visual Basic. Both of these findings are good news for most IS degree programs that struggle with providing sufficient course capacity to accommodate both high student demand and prudent course sequencing.

Students reported that they were more comfortable with the Visual Basic language when it was learned after or concurrently with another language. Thus, curriculum designers could either schedule to accommodate this observation or, alternatively, they might choose to put more 3GL fundamentals in their Visual Basic instruction.

Programming course instructors can observe there is no evidence to support student concerns of being disadvantaged in course grades if they did not take a prior programming course nor a concurrent course. The data here suggests that these perceived disadvantages are unfounded, though students report being more comfortable with Visual Basic when there is exposure to another programming course. Similarly, syllabi for programming courses can draw on this self-reported student data to affirm (and announce) what instructors have long known: course attendance significantly contributes to better course grades for all three languages.

We believe that future research initiatives should replicate the multi-semester, longitudinal design employed here to minimize one-semester class anomalies. Beyond the sequencing of the current languages, curriculum designers would benefit from empirically supported theoretical models that provide guidance regarding how to integrate new programming courses into their curriculum. Research on specific sequences and combinations of concurrent courses could give empirical insight for improved curriculum design. Given the enormous educational resources expended upon programming language instruction, even small improvements could have large effects on the hundreds of thousands of students in programming courses.

Four initiatives could define a basis for future research on the efficacy of curriculum sequence and concurrency.

First, future studies should also examine how courses in the Java programming language relate to sequence and concurrency with other courses. We would expect its results to be similar to courses in C, but evidence is needed to support this assertion.

Second, the results reported here could be affected by individual differences. Research designs that controlled for students with high or low aptitudes for learning programming languages could be especially insightful.

Third, other course domains in the IS curriculum merit similar attention. For example, what are the implications of sequence and concurrency for courses in traditional Systems Analysis and Design and Object-Oriented Systems Analysis and Design using Unified Modeling Language (UML)?

Finally, research designs that span universities could improve the generalizability of the results. Multi-university studies could help rule out issues related to specific instructors or local school culture.

Editors note: This article was received on Nov. 12, 2000 and was published on January 26, 2001.

REFERENCES

Campbell, D.T., and Stanley, J.C. (1963) *Experimental and Quasi-Experimental Designs for Research*. Boston: Houghton Mifflin Company

Couger, J.D., et al (1995) "IS '95: Guideline for Undergraduate IS Curriculum," *MIS Quarterly*, (19)3, pp. 341-359.

Davis, G., et al. (1997) "IS '97: Guideline for Undergraduate IS Curriculum,", in K. Kumar and J.I. DeGross(eds.) *Proceedings of the Eighteenth International Conference on InformationSystems,* New York, NY: ACM.

FOLDOC, Free Online Dictionary Of Computing, http://www.foldoc.org (Current 8 November 2000).

Manns, M., and Carlson, D. (1992) "Retraining Procedural Programmers: A Case Against 'Unlearning', *Object-Oriented Systems and LanguageApplications Conference*, pp. 131-133.

Powell, A. L (1997) "Programming Course Sequence and Prior Experience of Programming Languages: Do they Affect Students' Grades?" in J.N.D. Gupta (editor) *Proceedings of the Third AIS Americas Conference on Information Systems,* Atlanta GA: Association for Information Systems, pp. 771-773.

Rosson, M., and Alpert, S. (1990) "The Cognitive Consequences of Object-Oriented Design," *Human Computer Interaction*, pp. 360-378.

Webster (1993) *Webster's New Collegiate Dictionary*, Springfield, MA: Merriam Webster, p. 940

APPENDIX

PROGRAMMING CLASS SURVEY

This survey is designed to find your reactions to the courses which teach programming languages. Your responses to this survey will help us in future curriculum design and tailoring the courses to meet the desires of the students. Your responses will remain confidential and your individual answers will never be revealed to anyone.

Major _______ Current Year (e.g., Fr., So, etc.)_____ Appx GPA _____ __Major GPA ______

Circle the answer which best describes your response to the question **about this course**:

1. Your estimated letter grade in this course is: ________________

2. Describe your prior knowledge/skill of the language before taking this course:

1………………2……………………..3……………………..4………………….5………………….6………………………7
No Knowledge Moderate Skills Very Familiar

3. Describe your current level of comfort with the programming language taught in this course:

1………………2……………………..3……………………..4………………….5………………….6………………………7
Extremely Uncomfortable Neutral ExtremelyComfortable

4. Describe your frequency of attendance at regular class sessions:

1………………2……………………..3……………………..4………………….5………………….6………………………7
Never Half of the sessions All sessions

5. How interesting did you find this subject?

1………………2……………………..3……………………..4………………….5………………….6………………………7
Not at all interesting Moderately Interesting Very Interesting

6. Describe your desire to learn more about the language taught in this class:

1………………2……………………..3……………………..4………………….5………………….6………………………7
No Desire Some Desire Large Desire

7. How difficult did you find the course material?

1………………2……………………..3……………………..4………………….5………………….6………………………7
Very Difficult Moderately Difficult Not at all Difficult

8. How difficult did you find the course projects/assignments?

1………………2……………………..3……………………..4………………….5………………….6………………………7
Very Difficult Moderately Difficult Not at all Difficult

9. How difficult did you find the course examinations?

1………………2……………………..3……………………..4…………………5…………………6……………………..7
Very Difficult Moderately Difficult Not at all Difficult

If this is the only collegiate programming course you have **ever** taken, you are done with the survey. Please return the survey to the instructor.

If you have taken or are currently taking other collegiate programming courses, please continue.

10. Please list all the collegiate programming courses you have taken in the past or are currently taking **in the order that you took the courses**, noting the semester and the year you took the course. Then **rank order** the courses by difficulty level, noting the hardest with a 1, the second hardest with a 2, etc. **Include this course**. The first one is listed solely as an example:

University	School/College	Course No.	Semester/Year	Grade	Difficulty
MY U.	BUSINESS	X999	FALL 1996	B+	1

Please write on the back of this sheet if you need more space.

10a. Please complete a questionnaire about each of these other courses, as provided to you by the instructor.

11. Now please list the courses **in the order you wished you had taken them.** Number the list starting with 1, and if you wished to take the courses simultaneously, give them the same number in your list.

Number	Course Number

ABOUT THE AUTHORS

Andrew Urbaczewski is Assistant Professor of MIS at Washington State University. He holds a B.S. from the University of Tennessee, an MBA from West Virginia University, and a Ph.D. from Indiana University. His current research interests include IS education, electronic commerce, and electronic monitoring.

Bradley C. Wheeler is Associate Professor of MIS and a British American Tobacco Faculty Fellow for Technology-Supported Learning and Knowledge at Indiana University. His research addresses the use of digital networks for human collaboration, learning, and commerce. He currently teaches MBA and International Executive Education e-business courses.

Communications of the Association for Information Systems

ISSN: 1529-3181

EDITOR
Paul Gray
Claremont Graduate University