Hybrid Content Delivery and Learning Styles in a Computer Programming Course

Melissa Seward Yale, Deborah Bennett, Cordelia Brown, Guangwei Zhu, and Yung-Hsiang Lu
Purdue University, myale@purdue.edu, bennett@purdue.edu, brown83@purdue.edu,
guangwei@purdue.edu, yunglu@purdue.edu

***Abstract* - An experimental course design was implemented in a senior-level computer programming course at Purdue University. The course used hybrid content delivery, including online lectures and interactive problem-solving laboratory sessions. This hybrid design is a follow-on to previous hardware courses implementing a similar design with "directed problem solving" laboratories. This paper describes the course and its three major components: online lectures, interactive laboratories, and online discussions, along with students' responses to the effectiveness of various aspects of the course on their learning. Students' learning style preferences are compared with their achievement as reflected in five programming assignments. The programming assignments provide hands-on, practical experience for students. Each assignment consists of the development of a game in either C++ or Java. This paper suggests that the hybrid format is an effective way of learning for all students and provides an instructional environment conducive to varying learning styles. There is no strong indication that one style can outperform peers in this programming class.**

Index Terms – hybrid course delivery, interactive laboratories, learning styles, online lectures

INTRODUCTION

With the influx of technological advances in computer software, programming, debugging, and trouble-shooting are essential skills for computer engineering students. Traditional computer programming courses include syntax and logical thinking, but deeper problem-solving skills are not always explicitly developed [1]. However, coding skills, along with higher-level problem-solving ability is now necessary. The traditional course included a lecture component and supervised laboratory assignment, where students emulate similar programming to the lecture material. This method may be ineffective in teaching computer programming in today's technologically advancing software programs and processes. In the traditional course at Purdue University, the complexity of the material is difficult to explain and demonstrate in the typical 50-minute class period, three times a week. Compounding the problem, students enter programming courses with a wide range of programming experience. Some start programming in elementary schools and others have their first programming course in college. It can be difficult to engage every student and ensure that all the necessary material is covered. The traditional programming class does not have sufficient opportunities for students to have practical hands-on programming experience and develop skills in trouble-shooting.

A hybrid course design was developed for a sophomore- and junior-level hardware computer engineering course at Purdue University [2]. The hybrid design incorporated online lectures and mandatory "directed problem solving" laboratory sessions. These "directed problem solving" sessions fostered more face-to-face hands-on interaction between instructors and students and peer-to-peer. Students were able to self-select the experimental "directed problem solving" sections or traditional lecture and laboratory sections. To assist students in making this decision, the instructors used the Index of Learning Styles [3]. The Index of Learning Styles was developed to assess preferences on four dimensions (active/reflective, sensing/intuitive, visual/verbal, and sequential/global) of a learning style model formulated by Richard M. Felder and Linda K. Silverman [4]. The instrument was developed by Richard M. Felder and Barbara A. Soloman of North Carolina State University. It has been extensively administered to engineering students. According to Felder & Brent [5], traditional general engineering courses focus on theory and modeling versus practical and demonstrative experience, traditional instruction also includes more verbal explanations and lecturing, and convey information primarily through lectures and reading. The hybrid course design attempts to incorporate varying instructional methods and more hands-on practical applications of the material. Although many learners are capable of adapting to their instructional environment, instructional methods that modify traditional techniques to accommodate the majority of the students' learning styles should improve the overall educational experience. Students in the "directed problem solving" sections of the hardware course did report positive effectiveness of the hybrid course on their learning experience.

In Fall 2007, a modification of this hybrid design was adopted in a senior-level computer programming course for computer engineers [6]. Brown, Meyer, and Lu summarize students' initial responses to a course evaluation survey and found student responses to be positive towards the format and constructive feedback was used to improve the course for Fall 2008. This paper furthers the study by Brown, Meyer, and Lu by comparing specific learning styles to various assignments in the course.

The main objective of the course, ECE 462: Object-Oriented Programming in C++ and Java, is to educate better software developers with solid understanding of the concepts, proficiency in programming tools, and the ability to read and write non-trivial programs in C++ and Java. After taking the course, students should be able to develop object-oriented programs of moderate complexity from scratch or enhance existing programs. The course explains the concepts of object-oriented design, encapsulation, inheritance, exception, function and operator overloading, container classes, graphical user interface, networking, and concurrent programming using threads. Students should have Advanced C Programming (ECE 264) as a prerequisite.

The course has three major hybrid design components; on-demand lectures, hands-on laboratories, and online discussion forums. The hybrid course design permits the instructor to create quality instructional lectures and demonstrations to accommodate student's skill levels and learning styles. All lectures were recorded in advance and available on-demand using the popular Adobe flash format. The lectures include (1) explanation of essential concepts about object-oriented design and programming, (2) syntax of C++ and Java, and (3) sample programs and their outputs. Most graphical user interfaces (GUI) use object-oriented programming; hence, the outputs consist of user interfaces, graphics, and animation. (4) Comparisons of similar programs and their outputs. The comparisons include both correctness and performance. (5) Demonstrations of programming tools, including unified modeling language (UML) editor, debugger, version control, performance profiling, and test coverage. The lectures were recorded using Camtasia Studio. The videos were organized based on topics and recorded in segments less than 20 minutes long for easier access to particular sections of the lecture. Students could learn the material at their own pace with the ability to pause the videos to work on assignments or rewind certain sections to review the material as needed. Each video concludes with a short quiz reviewing the important concepts of the lecture. The quizzes include multiple-choice and short-answer questions. The students also receive immediate feedback when they enter their answers.

During the scheduled lecture time (50-minute sessions, three times a week), the instructor and the teaching assistant were available in the laboratory to assist students. There are five programming assignments; each assignment took 15-25 hours. All five programming assignments are related to interactive computer games. Computer games have been successful in teaching computer and engineering [7-10]. A computer game usually requires the following components; responses to user events such as moving the mouse cursor or pressing a key, active update of graphics, analytic geometry to detect collisions of moving objects, and sound effects. One assignment is a game played by two people using the Internet. Thus, the students will also learn how to use TCP/IP sockets. Additionally, ten exercises required students to write smaller programs or learn new tools, including Netbeans, Eclipse, Qt, version control, DDD debugger, GNU performance profiler and test coverage. Each exercise took about one hour and it was explained in a short video clip separate from the lectures. During the laboratory sessions, students had the opportunity to watch lecture videos, complete the exercises or assignments, and most importantly, interact with the instructor, the teaching assistant, and each other.

Peer-to-peer discussion is an essential component for this course. Students exchanged their experiences and collaborated in an online discussion group. The professor and teaching assistant did monitor the discussions and added additional information as needed after peer collaboration and discussion ensued. In 2007, students posted 192 messages and the instructor posted 177 messages. The postings covered a wide range of topics, including questions about the lectures, answers to other students' questions, and programming techniques not covered in lectures. In 2008, the discussion was divided into four categories: lecture, exercises, assignments, and miscellaneous. Students posted 307 messages and the instructor posted 85 messages. The significantly lower percentage of the instructor's postings in 2008 suggests that students were learning from each other and the instructor was not the sole source of information.

Recall that modifications were made to the 2008 implementation of the hybrid design course based on feedback received in 2007. This paper seeks to further the analysis of the hybrid design started by Brown, Meyer, and Lu [6] by comparing students' programming skills observed in the five game programming assignments with their learning styles as determined by the Index of Learning Styles. Our major research question is: Which learning styles are best suited for the hybrid course design?

METHODS

There were 37 students who completed the course with a passing grade in Fall 2008. Most students who received failing grades did not pass the engineering ABET outcomes for the course, completed less than fifty percent of the assignments, and are not included in this study. Of the 37 students who completed the course, 22 completed the Index of Learning Styles.

The Index of Learning Styles is a 44-item forced-choice questionnaire that measures four dimensions of learning styles; Active/Reflective, Sensory/Intuitive, Visual/Verbal, and Sequential/Global. Each dimension is scored on a scale and is a continuous dimension, not either or. For example, if a student receives a score of 1-3 on a dimension, the student is fairly balanced with a mild inclination toward a learning style, a score of 5-7 means the student is moderately inclined, and a score of 9-11 means the student is strongly inclined to one particular style of learning. The scoring method used in this study was the same used by Felder and Spurlin [11], the total of one dimension was subtracted from the other dimension's total responses to obtain a final score between -11 and 11.

TABLE I

DIMENSIONS OF LEARNING STYLES

	Count	%
Active	10	45.5%
Reflective	12	54.5%
Intuitive	8	36.4%
Sensing	14	63.6%
Visual	20	90.9%
Verbal	2	9.1%
Global	6	27.3%
Sequential	16	72.7%

Table I displays the number of students in each of the four dimensions. The participants were split almost half active and half reflective learners, over half were sensing learners, and the majority were visual learners and sequential learners. These are similar proportions to other learning styles research studies of engineering students by Felder and Brent [5], except reflective learners were more prominent in our study. Correlation coefficients were computed for each learning style dimension.

The first lecture was the only mandatory session. The instructor gave an overview of the course, its design, assignments, and grading policy. Students also voluntarily completed a background information survey and the Index of Learning Styles. Twenty-two students who completed the forms on the first day also completed the course. Most of the analysis compares these students' learning styles with course assignment grades and their final score in the course.

The five programming assignments were graded on functionality, coding standards, documentation, architecture, and overall evaluation (user experience) [12]. Architecture and evaluation were only on the fifth programming assignment, Interactive 2-Player Network Game using Java or C++. Examples of the grading criteria from the various assignments are given in Table II. Bonus points were also possible on each programming assignment for advanced game design and conceptual understanding, however these were left out of the main analysis since most students did not participate in the bonus portion of the assignments.

Qualitative responses to the students' attitudes towards the course were also collected in a small group setting towards the end of the semester. All small group analysis was completed anonymously and thus cannot be compared with assignment scores or specific learning styles. However, responses will be described briefly in the results section to show overall student opinion of the course. Eight students participated in the small group analysis.

RESULTS

The five major components of the programming assignments were totaled from each assignment and then standardized for comparisons. Total grades on each assignment were also standardized across all categories. Correlation coefficients were analyzed separately for each of the four learning styles dimension. When an analysis of variance (one-way ANOVA) was computed, the only dimension that presented statistical differences between the two contrasting learning styles was the Active/Reflective dimension. Programming assignment five's final grade and programming functionality total showed statistical significance between Active and Reflective learners (both $p<0.01$, $df=21$). This statistical difference may support the idea that reflective learners transferred the conceptual understanding needed to program a more complex interactive 2-player network game and overall scored higher on programming functionality within each assignment. Figure I displays the difference of mean

TABLE II.
PROGRAMMING ASSIGNMENT GRADING EXAMPLES

Program Functionality (Assignments 1 & 2 - Two-player breakout game using Java and C++)

- (15 pts) The ball bounces correctly when hitting the brick(s). The hit brick is removed. You must consider the situation of the ball hitting a corner of a brick (a bounding box of any shape may cause the collision to be detected prior to the actual occurrence, thus you cannot use bounding-box approximation for collision detection between the ball and a brick). If a ball goes behind the paddle, the ball is lost, and scores and number of balls left are correspondingly updated. A new ball is then appears in the playground, going towards the opponent, and before it reaches a paddle, it cannot hit any brick (to avoid the situation that no one can claim the score of the hit bricks).
- (3 pts) When user drags the slider, the velocity of the ball should change accordingly, within the specific range.

Program Coding Standards (Assignment 4 - Pacman game using C++ and Qt)

- (3pt) The code is formatted uniformly (use the function in your IDE).
- (3pt) There should be explanatory comments to help readers understand your code. Give a brief description about the class at the beginning of each class file.

Program Documentation (Assignment 3 – Tetrix game using Java)

- (3pt) Explain how you migrated the application from Qt to Java -- what remains unchanged and what is rewritten or appended.
- (4pt) Analyze the game algorithms such as rotation, collision, eliminating lines, etc. with sufficient details.
- (2pt) Explain the structure of the status files for Save and Open functionality.

Object-Oriented Architecture (Assignment 5)

- (5pt) Inheritance and Polymorphism: You must have at least one class inheriting from another base class. Somewhere in the program, the base class is used instead of the derived classes to exploit polymorphism. Describe the implementation in your documentation.
- (5pt) Networking layer: You must separate the codes related to networking from main window and playfield. A local player and a network player should share some common interfaces.

Overall Program Evaluation (Assignment 5)

- (3pt) Project Complexity: Judged from the number of classes and interfaces in your project. 1 point for 1-5 classes/interfaces, 2 points for 6-10 c/i, 3 points for over 10 classes/interfaces. Classes/interfaces not involved in the functionality of the project do not count (judged by your documentation and class diagram).
- (3pt) Creativity in Game: Higher score for more originality in game design and lower score for more common game. (Particularly, Breakout, Tetris and Pac-man games are worth 1 point for this criterion.)

scores on the average score of all five programming assignments for each of the four learning style dimensions. The score is standardized from 0 to 1 and the bars display the standard deviation of the scores. When the programming assignments were broken down into the rubric components listed in Table II, there only seems to be a possible distinction between active and reflective learners, all other dimensions, sensing/intuitive, visual/verbal, and global/sequential, do not have a distinct pattern of performance. Reflective learners appear to have performed better in each grading component when compared to active learners.

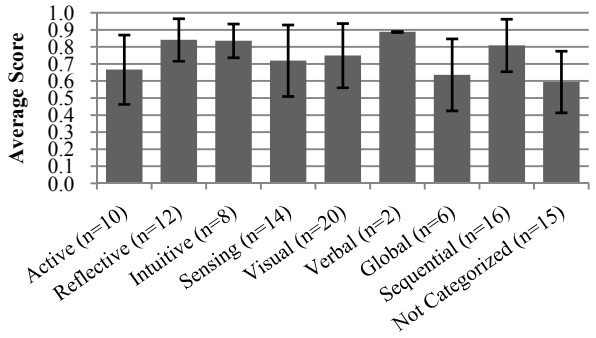

FIGURE I
MEAN DIFFERENCES OF LEARNING STYLES AND AVERAGE OF ALL PROGRAMMING ASSIGNMENTS.

Since there was not a linear relationship between learning style dimensions and scores, Spearman's rho, a nonparametric correlation coefficient was computed for each dimension and the results are reported in Table III. Each dimension is scaled from -11 to 11, below zero corresponds to Reflective, Intuitive, Verbal, and Global and respectively, above zero corresponds to Active, Sensing, Visual, and Sequential. If the correlation coefficient below is negative it corresponds towards that dimension of the scale. For example, Functionality has a rho = -0.55, meaning students had higher functionality scores if they were categorized as Reflective versus students categorized as Active.

When asked about the effectiveness of the three main components on their learning experience, students responded positively to the hybrid course design. On a five-point rating scale from very effective to very ineffective, no students rated the components as ineffective. All students (n=7) stated that the online lectures were either effective (n=4) or had no effect (n=3) on their experience. Two students stated that the interactive laboratories and programming assignments were very effective and four claimed them to be effective, one student did not have a response. As for the online discussions, two students rated it very effective and the remaining five as effective.

TABLE III.
CORRELATION COEFFICIENTS OF
LEARNING STYLES AND COURSE ACHIEVEMENT SCORES

	Active/ Reflective	Sensing/ Intuitive	Visual/ Verbal	Sequential/ Global	N
Architecture	-0.48*	-0.28	0.01	-0.23	19
Evaluation	-0.41	-0.10	0.00	-0.18	19
Functionality	-0.55**	-0.28	-0.07	0.43*	22
Documentation	-0.42	0.01	-0.16	0.42	22
Coding Standards	-0.19	0.07	-0.08	0.33	22
PA 1	-0.65**	-0.13	0.01	-0.02	22
PA 2	-0.54**	0.03	-0.11	0.36	22
PA 3	-0.43	-0.44	-0.08	0.22	20
PA 4	-0.49*	-0.04	0.00	0.49	19
PA 5	-0.68**	-0.28	-0.05	0.24	19
Final Score	-0.41	-0.13	0.18	0.33	22

*p≤0.05 **p≤0.01; PA=Programming Assignment

SUMMARY

We present a study on hybrid delivery of a programming course for computer engineers. Students watch online videos and interact with the instructor and the teaching assistant in laboratories. Students are encouraged to learn at their own pace and the instructional methods attempt to teach all learning styles. Analysis of the relationships between different learning styles and the students' accomplishments in the programming assignments show that there is no distinction between learning styles and their achievement in the course. Our study suggests that this hybrid format is appropriate for students of different learning styles. All students participating in this study indicated this hybrid approach provided an effective way of learning, through the online lectures, interactive laboratories, and online discussions. Future implementations of the course will continue to improve on the instructional videos, assignments, and use of technology. Also, additional studies will be conducted concerning online examinations and other online formats.

LIMITATIONS

Ideally, all students in the class would have participated in the Index of Learning Styles, but in our study that was not the case. The majority of the students who took the learning styles index were the top performers in the course overall. For example, 20 students received over 80% in the course, 17 of those students also took the learning styles index. This may have created some bias in the results, because the majority of students in our study were high performers overall. This possible bias could be deterred by the addition of more students. A larger n would also increase the statistical power of the results.

ACKNOWLEDGMENT

This study is supported in part by NSF CNS 0722212 and a Purdue Digital Content Development Grant. Any opinions, findings, and conclusions or recommendations are those of the authors and do not necessarily reflect the views of the sponsors.

REFERENCES

[1] Falkner, K, & Palmer, E, "Developing authentic problem solving skills in introductory computing classes", Proceedings of the 40th ACM technical symposium on Computer science education.

[2] Brown, C., & Meyers, D., "Experimental hybrid courses that combine online content delivery with face-to-face collaborative problem solving", Conference proceedings from American Society of Engineering Education, 2007.

[3] Soloman, B. A., & Felder, R. M., "Index of learning styles", http://www.engr.ncsu.edu/learningstyles/ilsweb.html.

[4] Felder, R. M., & Silverman, L. K., "Learning and teaching styles in engineering education", *Engineering Education*, 78(7), 1988, pp. 674-681. {www.ncsu.edu/effective_teaching/Papers/LS-1988.pdf}

[5] Felder, R. M., & Brent, R., "Understanding student differences", *Journal of Engineering Education, 94* (1), 2005, pp. 57-72. {http://www4.ncsu.edu/unity/lockers/users/f/felder/public/Papers/Understanding_Differences.pdf}

[6] Brown, C., Lu, Y., Meyer, D., & Johnson, M., "Hybrid content delivery: Online lectures and interactive lab assignments", American Society for Engineering Education Annual Conference Proceedings, 2008.

[7] Cliburn, D. C., & Miller, S. M., "What makes a "good" game programming assignment?" *Journal for Computing Sciences in Colleges*, 23, 4, April 2008, pp. 201-207.

[8] Aguilera, M. D., & Mendiz, A. "Video games and education", *ACM Computers in Entertainment*, 1, 1, October 2003.

[9] Gestwicki, P. & Sun, F., "Teaching design patterns through computer game development. *ACM Journal on Educational Resources in Computing*, 8, 1, March 2008.

[10] Mayo, M. J., "Games for Science and Engineering Education", *Communication of the ACM*, 50, 7, July 2007, pp. 31–36.

[11] Felder, R. M., & Spurlin, J., "Applications, reliability, and validity of the Index of Learning Styles", *International Journal of Engineering Education*, 21 (1), 2005, pp. 103-112. {http://www4.ncsu.edu/unity/lockers/users/f/felder/public/ILSdir/ILS_Validation(IJEE).pdf}

[12] Lu, Y. "ECE 462: Object-oriented programming in C++ and Java", {https://engineering.purdue.edu/OOSD}.

AUTHOR INFORMATION

Melissa Seward Yale, Graduate Research Assistant, Department of Educational Studies, Purdue University, myale@purdue.edu.

Deborah Bennett, Associate Professor Department of Educational Studies, Purdue University, bennett@purdue.edu

Cordelia Brown, Assistant Professor, School of Electrical and Computer Engineering, Purdue University, brown83@purdue.edu

Guangwei Zhu, Graduate Teaching Assistant, School of Electrical and Computer Engineering, Purdue University, guangwei@purdue.edu

Yung-Hsiang Lu, Associate Professor, School of Electrical and Computer Engineering, Purdue University, yunglu@purdue.edu