Enhancing STEM retention and graduation rate by incorporating innovative teaching strategies in selected STEM introductory courses

Dr. Nikunja Swain P.E., South Carolina State University

Dr. Swain is currently a Professor at the South Carolina State University. Dr. Swain has 25+ years of experience as an engineer and educator. He has more than 50 publications in journals and conference proceedings, has procured research and development grants from the NSF, NASA, DOT, DOD, and DOE and reviewed number of books on computer related areas. He is also a reviewer for ACM Computing Reviews, IJAMT, CIT, ASEE, and other conferences and journals. He is a registered Professional Engineer (PE) in South Carolina and ETAC of ABET reviewer for Electrical Engineering Technology and Computer Engineering Technology.

Dr. Biswajit Biswal, South Carolina State University

Biswajit Biswal, Ph.D.

Biswal is working as Assistant Professor of Computer Science at South Carolina State University, Orangeburg, SC, USA since January 2017. He holds Ph.D. in Computer and Information Systems Engineering from Tennessee State University, M.S. in Electrical Engineering form NYU Tondon School of Engineering, and B.E. in Medical Electronics Engineering from India. His research interests are machine learning, data mining, cyber security, cloud computing, RF signal detection (Drones), IOT, and big data analysis. He has more than 10 technical papers published in conferences and journals. He is also a member of IEEE.

Dr. Eugene Kennedy, Louisiana State University

Dr. Eugene Kennedy is an associate professor of Educational Research Methodology in the School of Education at Louisiana State University. He specializes in STEM education and research methods.

Enhancing STEM retention and graduation rate by incorporating innovative teaching strategies in selected STEM introductory courses

Abstract

Gate-keeping courses provide students with their first and formal exposure to a deep understanding of science. Such courses influence students' decision to pursue STEM education and continue their college experience. Our records indicate that the many STEM students perform poorly or marginally in the introductory required courses and decide to change their major to non-STEM degree programs. One way to address this is using active learning techniques.

The objective of this paper is to describe our experiences with the use of few of the active learning techniques in introductory computer programming courses offered in our Computer Science Program. One of these programming courses are required of all computer science majors and other course is usually taken by engineering, technology and science majors. The findings presented in this paper may be used by interested parties involved in STEM curriculum.

Introduction

The benefits of active learning have been supported time and again in the literature [1, 2, 3, 4, 5, 6, 7, 8, 9]. By comparing student learning gains in introductory physics courses, Richard Hake was able to show that interactive courses were over two times as effective in promoting conceptual understanding as compared to traditional ones [6]. Freeman et al. reported results from 225 studies across STEM disciplines, comparing traditional lecture to active learning [5]. In general, students' average exam scores were shown to improve by around 6% in active learning classes. Additionally, students involved in traditional lecture were found to be 1.5 times more likely to fail as compared to those in classes with significant active learning. Some of the active learning techniques are peer review, flipped classrooms, hands-on technology, and cooperative group problem solving. Here is a brief description of these methods [10].

In "peer review", students are asked to complete an individual homework assignment or short paper. On the day the assignment is due, students submit one copy to the instructor to be graded and one copy to their partner. Each student then takes their partner's work and, depending on the nature of the assignment, gives critical feedback, and corrects mistakes in content and/or grammar.

In the "flipped classroom", class time is devoted to engaged learning. Students are doing the "homework" (practice, application, and analysis of concepts) in class, often in collaboration with peers, and they can get help from their instructor and from peers as their questions arise. Therefore, students are expected to gain first exposure to concepts through readings or by watching videos before class, and they are held accountable for that pre-class work to ensure they prepare.

In “Hands-on technology”, students use technology such as simulation programs to get a deeper understanding of course concepts. For instance, students might use simulation software to design a simple device or use a statistical package for regression analysis.

In “Cooperative Groups in Class (Informal Groups, Triad Groups, etc.)”, Pose a question for each cooperative group while you circulate around the room answering questions, asking further questions, and keeping the groups on task. After allowing time for group discussion, ask students to share their discussion points with the rest of the class.

Introductory programming courses selected for Active Learning at University X

Introductory Programming Courses

At university X, computer science majors and mathematics majors are required to complete Programming I course with at least a C grade in the second semester of their freshman year. This course is usually taught using Java Programming Language. Students from other disciplines may take Introduction to Programming I course (CS 161). This course is taught using C++ programming language. The minimum passing grade for this course is C grade. The catalog description for both courses are shown below:

CS 160. Programming I 4(2, 2). This is the first programming course in the Computer Science sequence. It introduces students to programming with a structured programming language. Emphasis is on problem solving methods and algorithm development; definition of language syntax and semantics; and development of ability to apply concepts by designing coding, debugging, documenting, and executing programs. Topics include data types, variables, assignment, control structures (branching and looping). This course involves two hours of lecture and two hours of structured laboratory each week. Prerequisites: CS 151 or consent of instructor (F, S)

CS 161. Introduction to Programming. 3(2,1). An introduction to programming with a structured language on a standard computer system. Currently, we use C++ language and the UNIX operating system; but the choice of language and operating system depend on availability and currency. Emphasis is on understanding the various programming concepts. Some of the programming concepts include syntax, semantics, declarations, variables, input/output, formatting, selection, loops, subprograms, documentation, software engineering, and scope. Students apply those concepts by writing simple programs in the given language. This course involves two hours of lecture and one hour of structured laboratory each week. Prerequisite: None. (F, S)

Active Learning methods used

Flipped classroom and Hands-on technology were the two primary methods used in both courses. Students were required outside preparation of the content such as reading or watching videos on the subject matter before class. Videos were collected from variety of freely available resources on the web and modified to fit the course needs as needed. For the Hands-on technology, students spent most of their time in individual or group problem solving, and they used a visual programming language called RAPTOR with Java/C++ programming.

A Brief Description of the visual programming tool RAPTOR

The fundamental building blocks of any program are variables, sequence (input, output, assignment), selection (case, condition/Boolean, if-then-else), iteration (for, while, do while), arrays, file I/O, and functional decomposition (functions, procedures) and a novice programmer must have some understanding of few of these concepts. The visual programming tools can be effective and helpful tools to aid the novice programmers to learn these concepts.

Majority of the visualization tools are based on flowcharts because flowcharts use a graphical representation to describe the detailed logic of a process or set of rules. Flowcharts can be easily understood with little or no prior training. This small learning curve is due in part to the simplicity of their notation; any program can be expressed using only five basic flowcharting symbols as shown in Figure 1 below:

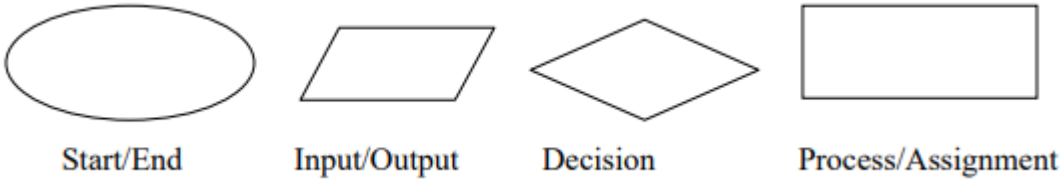

Figure 1 – Basic Flowchart Symbols

Flowcharts are ideal for modeling and visualization of small programs usually encountered by novice programmers. It allows the novice to envisage the flow of execution, the semantics of the conditional structures (i.e. if, if else, while, for etc.) and how the individual pieces of a program interact to form higher level concepts, whilst limiting the syntactic overheads of a programming language. Dynamic flowcharts are used to animate the dynamic nature of a computer program, its flow of execution, the changing state of program variables and the interactivity between program components. A dynamic flowchart also has better cognitive efficiency, as the viewer does not need to manually keep track of the state of program data, thus minimizing the amount of information the learner must hold in working memory or write down. By obeying the key rule of structured programming, that every structure has only one entry and exit point, flowcharts can easily communicate a well-defined and structured program. Subsequently, the transition from flowchart to structured code will be more apparent, requiring less reinterpretation. This should make the novice's transition from problem specification through to resultant structured code much simpler and more cognitively efficient.

Features of RAPTOR Visual Programming Tool Features of RAPTOR (11,12,13)

RAPTOR is a free visual programming tool developed by computer science faculty at the United States Air Force Academy. It is developed using C and Ada and freely available. Some of the features of RAPTOR are the following: Some of the features of RAPTOR are:

- The RAPTOR development environment minimizes the amount of syntax the student must learn to write correct program instructions.

• The RAPTOR development environment is visual. RAPTOR programs are diagrams (directed graphs) that can be executed one symbol at a time. This will help students to follow the flow of instruction execution in RAPTOR programs.

• RAPTOR error messages are designed to be more readily understandable by beginning programmers.

When students are learning to develop algorithms, they very often spend more time dealing with issues of syntax rather than solving the problem. Additionally, the textual nature of most programming environments works against the learning style of most students. RAPTOR is a visual programming environment, designed specifically to help students envision their algorithms and avoid syntactic baggage. RAPTOR programs are created visually and can be executed visually by tracing the execution through the program. Required syntax is kept to a minimum. Students preferred expressing their algorithms visually and were more successful creating algorithms using RAPTOR than using a traditional language or writing flowcharts. RAPTOR has 6 basic statements: Input, Output, Assignment, Call, Selection, and Loop. Each of these statements is indicated by a different symbol in RAPTOR as shown below in Figure 2.

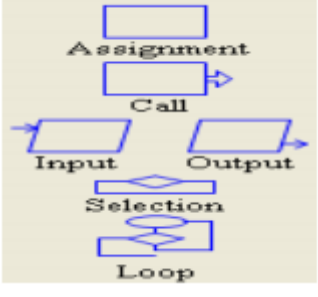

Figure 2 – Symbols in Raptor

Assessment and Evaluation

The assessment and evaluation of students in these courses were done by an external evaluator face-to-face interview and by an online survey. The instructors for these courses were also evaluated by the external evaluator for their experiences with active learning. The findings are presented in the following tables:

Table 1 – Summary of Student responses in CS 160 course (N = 7)

Questions	**Responses**
Overall, how would you grade (A, B, C, D, F) this course?	Most respondents (4 of 7) gave the course a grade of A. They noted the usefulness of the material and the structure that promoted interactions with classmates.
What aspects of the course helped you learn the material?	Respondents to this item noted hands-on activities and interactions with classmates and

	the instructor as being especially useful.
What aspects of the course were not particularly useful with respect to helping you learn the material?	Respondents to this item typically indicated that all aspects of the course were useful.
What things would you suggest as a way of improving the effectiveness of this course for students?	The responses to this item were evenly split between those who did not offer any suggestions for improvements and those who wanted more hands-on experiences
Did this course impact your interests in future computer science courses?	Six of the seven respondents indicated that the course did have a positive impact on their interests in taking future courses in computer science. The comments touched on the importance of computer science and the enjoyment and interest in the subject created in the course.
Did this course impact your attitude towards computer science as a discipline?	Five of the seven respondents indicated that the course impacted their view of computer science as a discipline.

Table 2 – Summary of Student responses in CS 161 course (N = 9)

Questions	**Responses**
Overall, how would you grade (A, B, C, D, F) this course?	The nine respondents were evenly divided in assigning the course grades of A, B and C. Those that assigned a grade of A tended to comment on the quality of the teaching and structure of the course. Those that assigned grades of B or C commented on their anticipated grades and opportunities for hands-on experiences and explanations offered during class.
What aspects of the course helped you learn the material?	Respondents to this item noted the value of in-class practice and the highly organized PowerPoints and other documents.

What aspects of the course were not particularly useful with respect to helping you learn the material?	The most consistently mentioned responses to this item addressed the quizzes and homework assignments.
What things would you suggest as a way of improving the effectiveness of this course for students?	Most respondents noted hands-on activities and classroom interactions as ways of making the course more effective.
Did this course impact your interests in future computer science courses?	Five of the nine respondents indicated that the course did have a positive impact on their interests in taking future courses in computer science. They commented on the value of computing and enjoyment of learning programming.
Did this course impact your attitude towards computer science as a discipline?	Most respondents indicated that the course did have a positive impact on their attitude towards computer science.

Table 3 – Summary of Failing Grades (D & F) in CS 160 course

(Spring 2017 – Spring 2019)

Semester	# of students	Failing grades (D & F)	%	Teaching Method
Spring 2017	27	15	56%	Traditional
Fall 2017	22	4	18%	Traditional
Spring 2018	29	16	55%	Traditional
Fall 2018	12	2	17%	Traditional
Spring 2019	26	15	58%	Traditional
Fall 2019	13	0	0%	Active Learning (Pilot)

Table 4 – Summary of Failing Grades (D & F) in CS 161 course

(Spring 2017 – Spring 2019)

Semester	# of students	Failing grades (D & F)	%	Teaching Method
Spring 2017	13	1	8%	Traditional
Fall 2017	5	0	0%	Traditional
Spring 2018	15	0	0%	Traditional
Fall 2018	12	2	17%	Traditional
Spring 2019	6	1	17%	Traditional
Fall 2019	16	3	19%	Active Learning (Pilot)

Summary and Conclusions

As seen in Tables 1 and 2, the flipped classroom and hands-on technology active learning methods along with visual programming tool helped the students to understand the concepts in more detail. The RAPTOR program helped the students to enhance their logical and critical thinking skills. Students also indicated that the course has a positive impact towards computer science as discipline. Table 3 also indicated that there is a reduction rate of failing rates from Spring 2019 to Fall 2019. It seems as if the combination of active learning and visual tool RAPTOR may have contributed to this effort but for to draw any meaningful conclusion, we will need additional semester data. Results from Table 4 does not show any significant effect of active learning and visual tool RAPTOR on student grades. This may be since students are from disciplines other than computer science with little to no background in computing and lack of student participation in problem solving and group activities. This was mentioned by the instructor of CS 161 course in response to the external evaluator question on "Active Learning Strategy used in My Class - Most of the students did ignore my advice on working in groups (they were comfortable working their own)."

We are using flipped classroom and hands-on technology along with visual tool RAPTOR in CS 160 and CS 161 courses this semester, and we plan to continue using these in subsequent semesters. We are also in the process of developing custom material (Active learning, RAPTOR, and Java/C++ programming) for these two courses. These materials will be available to students and interested faculty free of charge one completed.

Acknowledgement

Current funding for this project has been provided by the National Science Foundation through award HRD-1912085 Additional resources have been provided by SCSU. The authors wish to acknowledge this support and thank NSF for this grant.

References

[1] T. Briggs, "Techniques for Active Learning in CS Courses," Journal of Computing Sciences in Colleges, Vol. 21, no. 2, 2005, pp. 156 – 165.

[2] Bonwell, C.C. & Eisen, J.A. (1991). Active Learning: Creating Excitement in the Classroom. School of Education and Human Development, George Washington University: Washington DC.

[3] FEison, J. (2010). Using active learning instructional strategies to create excitement and enhance learning. Jurnal Pendidikantentang Strategi Pembelajaran Aktif (Active Learning) Books, 2.

[4] Finelli, C.J., Nguyen, K., Demonbrun, M., Borrego, M., Prince, M., Husman, J., Henderson, C., Shekhar, P., & Waters, C. K. (2018). Reducing student resistance to active learning: Strategies for instructors. Journal of College Science Teaching, 47(5), 80-91.

[5] Freeman, S., Eddy, S. L., McDonough, M., Smith, M. K., Okoroafor, N., Jordt, H., & Wenderoth, M. P. (2014). Active learning increases student performance in science, engineering, and mathematics. Proceedings of the National Academy of Sciences, 111(23), 8410-8415.

[6] Hake, R. R. (1998). Interactive-engagement versus traditional methods: A six-thousand-student survey of mechanics test data for introductory physics courses. American journal of Physics, 66(1), 64-74.

[7] Kothiyal, A., Murthy, S., & Iyer, S. (2014). Think-pair-share in a large CS1 class. Proceedings of the 2014 Conference on Innovation & Technology in Computer Science Education - ITiCSE '14. doi:10.1145/2591708.2591739

[8] McCarthy, J. P., & Anderson, L. (2000). Active learning techniques versus traditional teaching styles: two experiments from history and political science. Innovative Higher Education, 24(4), 279-294.

[9] Prince, M. (2004). Does Active Learning Work? A Review of the Research. Journal of engineering education, 93(3), 223-231.

[10] Active Learning Techniques. Retrieved from http://crlt.umich.edu/sites/default/files/resource_files/Active%20Learning%20Continuum%20Techniques.pdf

[11] RAPTOR – Retrieved from http://raptor.martincarlisle.com/

[12] W. Brown, "Introduction to Programming with RAPTOR," Retrieved from http://www.scribd.com/doc/13826372/Introduction-to-Programming-With-RAPTOR

[13] Venit, S and Drake, E. (2011) – Prelude to Programming – Concepts and Design, 5th Edition, Addison-Wesley.