Visual Programming and Interactive Learning Based Dynamic Instructional Approaches to Teach an Introductory Programming Course

Md Mahmudur Rahman, Roshan Paudel
Computer Science Department
Morgan State University
Baltimore, Maryland, USA
md.rahman@morgan.edu, roshan.paudel@moragna.edu

***Abstract*— This Research to Practice Full Paper presents our experience in teaching an introductory programming course in Python by using a visual programming development environment based on flow-charts and active learning with an interactive eBook. The field of computer science education is always being challenged with the high attrition rates despite the ever growing industry demand for computing expertise. The lower rate of student retention is often associated with considerable dropout and failure rates in introductory programming courses during the freshmen year. The main challenge is getting students to write meaningful programs in a short time by focusing more on computational thinking principles and less on language details at that point in time. Nowadays, most students prefer to be engaged and discover course content through exploration, interaction, and collaboration that is relevant, useful, and fun compared to traditional blackboard-based lecturing styles. For evaluation of learning outcomes based on the quantifiable criteria with robust statistical analysis, eleven sections of the course over three semesters were considered. The initial evaluation of summative assessment and analysis of a survey result enable us to conclude that the proposed instructional approach increases student engagement, facilitate learning and contribute to the progress of students in this course.**

Keywords—Introductory programming, computational thinking, visual learning, interactive learning, eBook, active learning, assessment

I. INTRODUCTION

Computer programming is among the most challenging subjects in Computer Science (CS) curriculum for freshmen, particularly for those without any exposure to computing at the high school level. A large proportion of students are having difficulty getting through the programming concepts to understand how a static textual representation (source code) maps to a highly dynamic process (program execution). As a result, many students got overwhelmed by the breadth and depth of this very first course and ultimately failed and changed majors as a result [1]. Despite many attempts to make introductory programming languages easy to comprehend, many beginners still struggle with the syntax and semantics [2-8]. The failure rates in the first programming courses are as high as 30% [2] and most students drop out of the CS major after taking those. In a similar trend, we observed the highest drop rate in introductory programming courses over the years even though most of the students have had some prior experience with computers. The course evaluations indicate that most of them feel that programming is a challenging intellectual exercise. The failing rate is significant enough to be a matter of concern and worry. Hence, it is important for us to find a good strategy to recruit freshman students to CS courses and to keep them in the program once they are in.

Several problems are identified, such as lack of problem-solving and analytical thinking skills as weaker or underprepared student admitted to the program and lack of self-confidence due to poor socio-economic background for majority of the students in our school. In addition, the traditional blackboard-based lecturing styles in our classrooms are also insufficient in grasping the attention of these students and stimulating student interests. The programming language should right away allow a focus on computational principles as a tool to develop higher knowledge in CS so that students are able to write meaningful programs in a short time, whilst limiting the syntactic overheads [9,10]. The freshmen might be become better problem solvers and algorithmic thinkers at first by using a visual programming development environment based on flowcharts. A flowchart is a collection of connected graphic symbols, where each symbol represents a specific type of instruction to be executed based on connections between symbols [11]. Students can build simple procedural programs without learning the details of a language at the very beginning of the learning process. Research findings [3-6] in CS education also suggest that teaching computational thinking (CT) and problem solving skills before or alongside traditional programming yielded significant improvements in student performance in subsequent courses. There are also a large body of evidence supporting the idea that most students nowadays are visual learners who learn programming concept better through web-based visual, interactive, and collaborative

978-1-5386-1174-6/18/$31.00 ©2018 IEEE

learning instead of learning from traditional black board lecturing styles [7-9].

Also, the choice of what to teach first as a general programming language should eventually, influence many students' first impressions of CS. In recent years, Python has tended to be the language of choice for CS1 courses across many universities. Many findings [12,13] revealed that Python as a simple and expressive scripting and interactive programming language is easier for beginners, which provides an interactive, speedier, less overwhelming, and gentler introduction to the basic concepts of programming. However, only choosing the appropriate language may not drive engagement and promote success to students of the new digital generation experienced the world in which Facebook, Twitter, Google, iPhone's, iPad's, and countless other technological advances exist [14]. Nowadays, most students are accustomed to interactive and visual learning process as early as in elementary schools. They prefer to be engaged and discover course content through exploration, interaction, and collaboration that is relevant, active, instantly useful, and fun instead of being lectured for the duration of a class. Using appropriate teaching strategies and establishing a conducive and dynamic classroom environment an instructor can develop a positive learning experience that can successfully.

There is a significantly large number of articles published and techniques implemented from educational research community to address the issue in introductory programming courses [1-8]. Hence, instead of "reinventing the wheel", we implemented a combined instructional approach by infusing both visual and interactive learning. In this paper, we describe our experience of instructing an introductory CS course (Introduction to Computer Science I, COSC 111) in Python in the Spring 2017 semester and positive learning outcomes of infusing visual learning with a flow-chart based visual programming environment in RAPTOR [15] and interactive learning with the usage of an interactive book, zyBook [16]. In this study, eleven sections of COSC 111 were included over three semesters as control and experimental groups. Analysis showed a marginally significant difference ($F(1,5) = 5.87$; $p = 0.060$) between sections taught via the traditional method and sections taught with the proposed pedagogical innovation. We also found that there are statistically significant positive correlations in between uses of the interactive book (eBook) and performances of students in tests. To add to our understanding of what students were experiencing, we also administered a survey to students at the end of the course. Overall, it seems that the proposed pedagogical approaches have made a positive difference by increasing student motivation and engagement, and reducing failure rates.

II. Introducing a Flowchart-based Visual Programming Tool

To practice CT as a process of solving problems with a focus on algorithms, decomposition, abstractions, and pattern recognition based on data representation and data analysis, one must be able to express these thoughts and flows without relating them to an executable language at first. Otherwise, the abstraction will be of little use and the algorithms might end up poorly executed. Visual programs based on flowcharts allow students to visualize how programs work and develop algorithms in a more intuitive fashion. A flowchart is a collection of connected graphic symbols, where each symbol represents a specific type of instruction to be executed based on connections between symbols. The potential of visualization and how the use of dynamic structured flowcharts may be effective in aiding the conceptual understanding and problem-solving skills of novice programmers is discussed in detail in a doctoral dissertation, titled "Using Flowcharts, Code and Animation for Improved Comprehension and Ability in Novice Programming" by Dr. Andrew Scott [11]. The flow model greatly reduces syntactic complexity, as students can build simple procedural programs without learning the details of a language.

Being motivated, we have decided to use RAPTOR [15,17], which is a free flowchart-based programming environment designed at the United States Air Force Academy. The decision is based on the following criteria: 1) its visual development environment minimizes the amount of syntax students must learn to write correct program instructions 2) interface provides students with an intuitive means of adding sequential, selection, and repetition structures to their flowcharts, 3) it has an easy to use graphics library, built-in functions and procedures that allows students to create interesting programs using animations, draw graphics (including circles, boxes, lines, etc.), generate random numbers, perform trigonometric computations, etc. and finally good documentation and user guide is available in RAPTOR website [17].

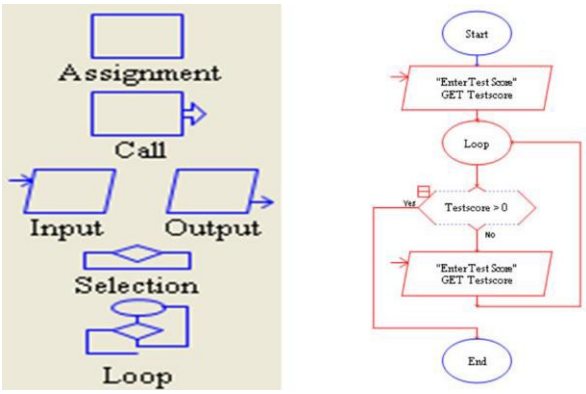

Figure 1: (a) Symbols in RAPTOR (b) Flowchart in RAPTOR

RAPTOR programs are created visually and can be executed visually by tracing the execution through the program. Students preferred expressing their algorithms visually, and were more successful creating algorithms using RAPTOR than using a traditional language or writing flowcharts. It has six basic statements***: Input, Output, Assignment, Call, Selection,*** and ***Loop***, where each of these statements is indicated by a different symbol in RAPTOR shown in Fig. 1 (a). **The** RAPTOR solution is independent of syntax and does not require detailed

understanding of loops, input/output and processes (Fig. 2 (b)). It takes less time to understand the concept since only 6 symbols are used to solve the problems, hence, it seems to be very suitable for teaching introductory CS courses before introducing to any general programming languages [15].

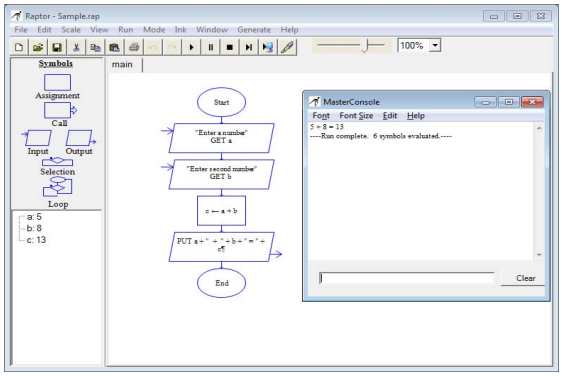

Figure 2: A snapshot of programming environment in RAPTOR

We have created several lesson plans and exercises revolving around the CT concepts (for example Fig. 2) and spent the entire first three weeks of our instruction out of fifteen weeks in RAPTOR before transitioning to Python programming.

III. Infusing Interactive Learning

A core challenge in introductory CS courses is getting students to understand how a static textual representation (source code) maps to a highly dynamic process (program execution)[18]. The program execution is typically illustrated using graphical PowerPoint lecture slides, which require a great deal of preparation effort, or by drawing diagrams on a whiteboard, which is tedious and error prone. Meanwhile, the real power of the web for learning—interactivity—has barely been tapped in our classes. In the past few decades, many program visualization tools have been created to assist instructors in this task [19]. However, using these visual tools along with other traditional instructional materials (e.g., textbooks, lecture notes, instructional videos) require the students to encounter too many different environments and that might lead to the kind of split-attention problem that can hurt students' learning. Recently, new kinds of integrated electronic textbooks (eBooks) [20,21] can provide a single interface for a wide variety of instructional materials. These new kinds of integrated electronic textbooks (eBooks) utilize the power of the web (HTML5, CSS, and JavaScript) with substantially less text than a traditional textbook and not a digital version of it, instead having numerous embedded responsive question sets, animations, interactive exercises, and with some built-in program editing and execution tools. The good thing is that students don't have to install anything, and they can read, edit, and run programs all from within the pages of these eBooks inside the browser and see immediately the results of their work.

To teach Python in COSC 111 (Spring 2017), we have used such an interactive book known as zyBook of Zyante's [16]. It is basically a web-native interactive content which consisted of question sets, animations, interactive tools, and embedded homework, so students can have effective learning experience outside of lecture. It also includes reorderable content, auto-generated problems of varying difficulty, and a student and instructor dashboard for monitoring activity. A recent study involved around 2,000 students at four introductory programming courses at three universities showed improvements in all aspects of student performance (quizzes, exams, projects) with strong statistical significance when static textbooks were replaced by the interactive zyBooks [22]. The book offers the following main features, which we found very effective for students learning:

Animation and Software visualization tool (Fig. 3): Students can step through code line-byline, forward or backward or all the way to the start or end of the program execution.

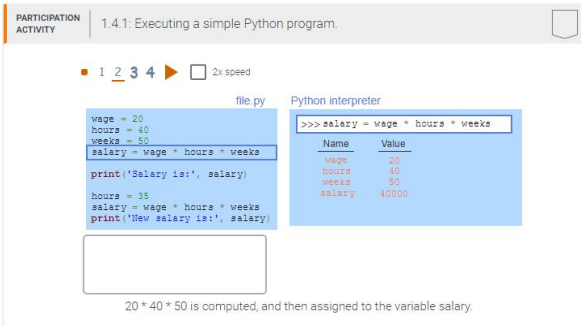

Figure 3: Snapshot of a Software visualization tool [16]

A program editing and execution area (Fig. 4). Students can execute examples, change them, and execute the updated code. They can also save and load their modified co

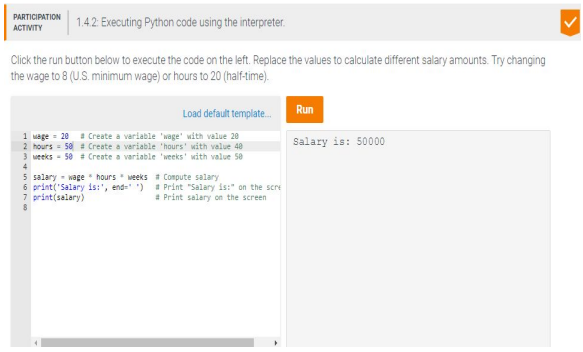

Figure 4: Program editing and execution area [16]

Several kinds of learning questions (effective form of explanatory text) in the forms of short answer, multiple choice, true/false, matching (Fig. 5) and more. If a user provides a wrong answer, a short answer question provides a hint (Fig. 6), while a multiple choice or true/false question explains why the answer is wrong. The user can attempt answering as many times as desired.

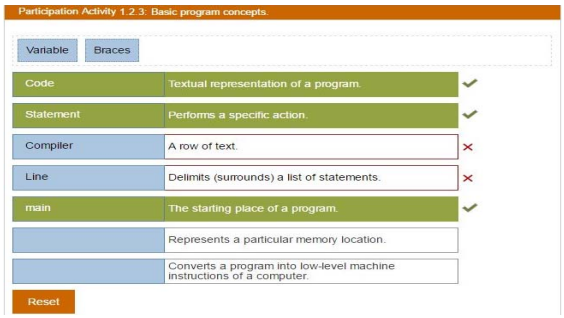

Figure 5: Learning questions as arranging rows correctly (matching) [16]

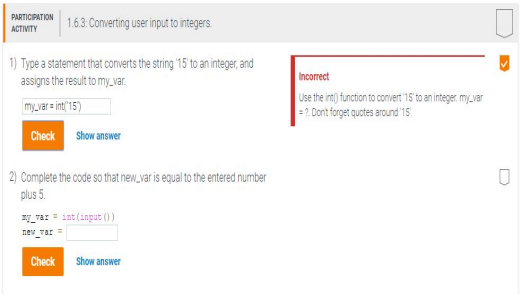

Figure 6: Learning questions as short answer [16]

A. *Our Approach of using zyBook*

In Spring' 2017, we offered instruction of COSC 111 in three different sections (with two instructors) of twenty students in each section with a total of sixty students. We recently (Fall'2016) transitioned from C++ to Python due to its good appeal and effectiveness as a programming language for the beginner. Students in all three sections were subscribed (based on a funding of the authors) to a zyBook, "Programming in Python 3" at the beginning of the semester. In the previous section, it demonstrates how the zyBook provides instructional information with creative exploration and opportunities to online code simulation on the same page that is impossible on the physical page. However, just because our eBook provides these opportunities does not mean that we know how students will use it in the context of a class.

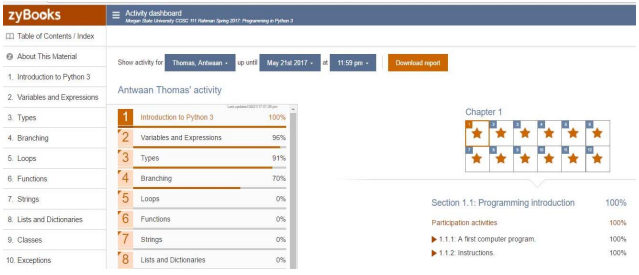

Figure 7: Activity dashboard of an individual student

To engage the students more actively with the book both in and out of the class, we assigned some course grade points (e.g., 15% of the course grade) for completing all the online exercises (interactive textbook activities, automatically logged when student does the activities while reading) by the given due dates. An instructor can easily observe the percentage of completion per student as well as the entire class in the activity dashboard at any point in time as shown in Fig. 7 and Fig. 8 respectively.

Another important instructional approach we followed is engaging students to do in-class exercises from the eBook for the last 15-20 minutes immediately after discussing a topic/concept in our 50 minutes lecture. It was previously found that students who are passive have a decline in concentration after 10-15 minutes in a 60-minute lecture [23].

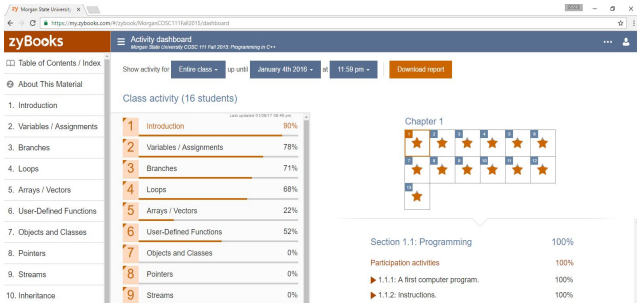

Figure 8: Activity dashboard of an entire section of COSC 111

The most important feature of our eBook is the immediate feedback that the students received by participating in both in and out of class activities, which allows them to see immediately the results of their work, including the errors reported and gives them the opportunity to adjust accordingly. In our approach, the students spend more time actively engaged in small group exercises and interactive demonstrations instead of listening to a traditional lecture. In this context, the value of active learning is realized to some extent in our classroom through interactive lecturing styles.

IV. Learning Outcomes and Result Evaluation

This work is exploratory in nature and the primary research question is: do the proposed techniques increase student engagement, facilitate learning, and contribute to the progress of students in the course? The ultimate success will be measured as 1) preparedness of COSC 111 students for consequent courses 2) the rate of students who stay as CS major (retention rate in CS) 3) the rate of students who declare their major/minor as CS after COSC 111 is completed. To address the primary research question and produce teaching recommendations, the proposed approaches are evaluated based on the quantifiable criteria with robust statistical analysis. We applied the proposed pedagogical approaches for the first time in the Spring'2017 semester to teach our introductory CS course, COSC 111. The traditional course instruction that has historically been used in the department are used as the control group so that students experiencing the new approaches as the experimental group can be compared to that of previous students within the department. We measured the effectiveness of the proposed instructional approaches through grade distributions, self-reports, and surveys of student attitudes toward programming.

For evaluation and result analysis, eleven sections of COSC 111 were included in this study over three semesters: Fall 2014, Fall 2016 and Spring 2017. A total of 230 students were enrolled in across these sections. Across the three semester included for this analysis, three different pedagogical approaches were used: (1) the traditional way that the course has been taught using C++ as the first programing language in the CS department at our University. (2) Using Python instead of C++, but with no other pedagogical innovations. (3) Using Python, but also implementing the proposed pedagogical approaches.

Table 1: Course Enrollment

		ENROLLMENT, INDIVIDUAL SECTIONS*					TERM TOTAL
F2014	Traditional (C++)	23	25	18	22		88
F2016	Python w/o Proposed Innovations	23	24	25	19		91
Sp2017	Python w/Proposed Innovations	17	17	17			51

Table 1 includes a summary of the course enrollment and identifies which teaching methods were used in which semesters.

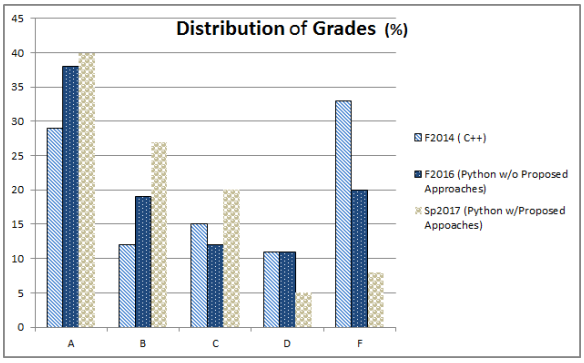

Figure 8: Comparison between Student's final grades in three different semesters

Due to faculty workload constraints, only end-of-semester grade distributions were available across the sections included here. We can initially observe in Fig. 8 that the proposed pedagogical techniques (Sp2017) reduced the failure rates from 44% in Fall 2014 to only 13% in Spring 2017 when we consider both 'D' and 'F' grades. Given differences in the number of sections and the enrollment across sections, grade distributions (Fig. 10) were converted to quality point scores and averaged. That is, for each section, the number of students who received an 'A' was multiplied by 4 quality points, the number who received a 'B' by 3 quality points, etc. Then, the total for each section was divided by the number of student who received a grade in the course. This equalized the scores for subsequent analysis.

Table 2: Quality Point Scores and One-Way ANOVA by Method

		Pedagogical Approach	Quality Point Scores by Section				Mean Quality Points Score
F2014	#1	Traditional (C++)	1.18	1.87	2.12	2.61	1.94*
F2016	#2	Python w/o Proposed Innovations	2.08	2.47	2.58	2.69	2.45
Sp2017	#3	Python w/Prpoosed Innovations	2.59	2.88	3.09		2.85*

*difference between methods #1 and #3: F(1,5) = 5.87; p = 0.06

Analysis (Table 2) showed a marginally significant difference between sections taught via the traditional method and sections taught with Python and the proposed pedagogical innovation (F (1,5) = 5.87; p = 0.060). Though grades are admittedly an indirect method of assessment, there are multiple sections represented here, and these quality point scores do shows a notable difference in performance across the pedagogical approaches.

Table 3: Student Self-Reports of Course Experiences (n = 35)

		Strongly Disagree		Disagree		Neutral		Agree		Strongly Agree		Mean[2]
		N	%	N	%	N	%	N	%	N	%	
PROGRAMMING COMPETENCE												
1	I began this course with little to no programming knowledge.	1	2.9	5	14.3	6	17.1	7	20.0	16	45.7	3.91
2	Having completed this course, I now feel that I could program anything in Python.	0	0	4	11.4	11	31.4	16	45.7	4	11.4	3.57
PLANS FOR FUTURE COMPUTER SCIENCE STUDY												
		Strongly Disagree		Disagree		Neutral		Agree		Strongly Agree		
		N	%	N	%	N	%	N	%	N	%	
3	This class reduced my interest in computer science.	14	40.0	13	37.1	6	17.1	1	2.9	1	2.9	1.91
4	As a result of my experiences in this class, I plan to take more computer science courses.	2	5.7	3	8.6	11	31.4	9	25.7	10	28.6	3.63
PREFERRED LEARNING METHODS AND MATERIALS												
		Strongly Disagree		Disagree		Neutral		Agree		Strongly Agree		
		N	%	N	%	N	%	N	%	N	%	
5	I prefer to read a textbook in hard copy rather than on a screen	11	31.4	9	25.7	6	17.1	6	17.1	3	8.6	2.46
6	If I were to take another computer science course, I would want a book similar to the online book used for this one	2	5.7	4	11.4	8	22.9	12	34.3	9	25.7	3.63
7	The online book helped me in understanding the programming concepts	1	2.9	3	8.6	6	17.1	15	42.9	10	28.6	3.86
8	The online code simulation and instant feedback on the eBook was helpful to me	1	2.9	4	11.4	1	2.9	15	42.9	14	40.0	4.06
9	Discussion and interaction in the classroom was important in clarifying many programming concepts	0	0	1	2.9	4	11.4	12	34.3	18	51.4	4.34
10	I devoted about the same amount of time to studying for this class as other courses	1	2.9	6	17.1	7	20.0	14	40.0	7	20.0	3.57
11	My methods of studying for this class was similar to what I use for other courses	6	17.7	9	25.7	9	25.7	6	17.7	5	14.3	2.86

To add to our understanding of what students were experiencing with our enhanced instructional approaches, we also administered a survey to students at the end of the course (Spring 2017 sections). A total of 35 students completed the paper survey, for 68.6% response rate. The five-point Likert scale survey (1 = strongly disagree; 2 = disagree; 3 = neutral; 4 = agree; 5 = strongly agree) included items focusing on their experience with programming prior to the course, their beliefs in their level of competence after the course, their preferred learning methods, and how their preparation for the COCS111 differed from their preparation for other courses. The complete survey and student responses are summarized in Table 3.

Results of the student survey showed that the majority of those who responded had little to no programming experience before they enrolled in COCS 111, and the felt capable of programming in Python because of completing the course. Most students agreed or strongly agreed that because of their experience in the COCS 111 course, they planned to take more computer science courses (54.3%). Regarding learning styles

and tools, most student disagreed or strongly disagreed that they'd prefer at hard copy textbook (57.1%), and most agreed or strongly agreed that they would want to use a similar online textbook in future computer science courses (60.0%), that the online book clarified programming concepts (71.5%), that the instant feedback that the online book provided was helpful (82.9%), that the class discussion and interaction were helpful (85.7%). Students were also asked how their method of preparing for COCS 111 differed from how they prepared for other courses. Up to two responses were coded from the students' open-ended responses. The most frequent strategies mentioned were working similar practice problems (mentioned by 19.4% of respondents), and using the zyBook (mentioned by over 14% of respondents).

Overall, it seems that the proposed pedagogical approaches have made a positive difference, despite the limitations to the available data to date. Based on the above studies, our preliminary results are very encouraging and suggest improvement in enabling higher learner engagement and more-effective learning in addition to increase motivation towards programming. It enables us to conclude that the proposed techniques increased student engagement, reduced failure rates and thereby increased retention. Our ultimate goal is to infuse CT and modern educational technologies throughout the CS1/CS2 courses to create an intellectual, active and engaging academic environment and to enhance student and teacher interaction.

V. CONCLUSION

Significantly high numbers of articles are published on techniques to teach introductory CS courses. Instead of "reinventing the wheel," we proposed to implement some already successful pedagogy in CS by introducing Python as the introductory programming language, incorporating visual and interactive learning in the classroom through using a web-based eBook and actively engaging students to stimulate students' interests instead of listening to a traditional lecture. We expect that our ongoing instructional approach would result in a faster acquisition of the needed skills and in producing more confidence in students in programming code development, positioning them well prepared for more advanced courses in the curriculum, and retaining them to continue in the CS program. The initial evaluation and data analysis showed that we are heading in a positive direction and the desired outcome will be far-reaching to meet the growing needs of the STEM employers beyond just increasing student's learning experience and motivation toward programming.

ACKNOWLEDGMENT

This research was supported by an NSF HBCU-UP Targeted Infusion Project (TIP) grant (Award No:1623335).

REFERENCES

[1] A. Forte and M. Guzdial, "Motivation and nonmajors in computer science: identifying discrete audiences for introductory courses". IEEE Transactions on Education, vol. 48(2), pp. 248–253, May 2005.

[2] R. Bryant, K. Sutner and M. Stehli., Introductory Computer Science Education at Carnegie Mellon University: A Deans' Perspective. Online: http://reports-archive.adm.cs.cmu.edu/anon/anon/home/ftp/2010/ CMU-CS-10-140.pdf , 2010.

[3] A. Pears, S. Seidman, L. Malmi, L. Mannila, E. Adams, J. Bennedsen, M. Devlin, and J. Paterson. A survey of literature on the teaching of introductory programming. SIGCSE Bull.,vpl. 39 pp. 204–223, December 2007.

[4] C. Schulte and J. Bennedsen. 2006. "What do teachers teach in introductory programming?," Proceedings of the second international workshop on Computing education research, pp. 17-28, 2006.

[6] S. Davies, J. A. Polack-Wahl and K. Anewalt, "A Snapshot of Current Practices in Teaching the Introductory Programming Sequence," in SIGCSE 2011, 2011.

[7] T. Koulouri, S. Lauria and R.D. Macredie, Teaching Introductory Programming: A Quantitative Evaluation of Different Approaches, ACM Transactions on Computing Education (TOCE), vol. 14 (4), February 2015.

[8] S. Judy, S. Simon, H. Margaret and L. Jan, Analysis of research into the teaching and learning of programming, In Proc. fifth international workshop on Computing education research workshop (ICER '09), pp. 93-104, 2009.

[9] J. J Lu and G. HL Fletcher. Thinking about computational thinking. In ACM SIGCSE Bulletin, vol. 41, pp. 260–264. ACM, 2009.

[10] J. M. Wing. Computational thinking. Communications of the ACM, vol. 49(3), pp. 33–35, 2006.

[11] A. Scott, "Using Flowcharts, Code and Animation for Improved Comprehension and Ability in Novice Programming", University of Glamorgan, UK. 2010.

[12] Richard J. Enbody, William F. Punch, Mark McCullen, Python CS1 as preparation for C++ CS2, Proceedings of the 40th ACM technical symposium on Computer science education, Chattanooga, TN, USA, March 04-07, 2009.

[13] P.J. Guo, "Python is Now the Most Popular Introductory Teaching Language at Top U.S. Universities", Online: http://cacm.acm.org/blogs/blog-cacm/176450-python-is-now-the-most-popular-introductory-teaching-language-at-top-us-universities/fulltext

[14] Understanding the digital generation: Teaching and learning in the new digital landscape. (2010). Book by Ian Jukes, Ted McCain, & Lee Crockett. Review by Douglas Smith. Thousand Oaks, CA: Corwin Press (SAGE),152 pp.

[15] M.C. Carlisle, "RAPTOR: a visual programming environment for teaching algorithmic problem solving," in Proc. SIGCSE 2005.

[16] zyBooks: Online: https://zybooks.zyante.com/#/home [Accessed 25 June. 2018].

[17] RAPTOR, Home page– http://raptor.martincarlisle.com/ [Accessed 25 June. 2018].

[18] P.J. Guo, Online Python Tutor: Embeddable Web-Based Program Visualization for CS Education. In Proceedings of the ACM Technical Symposium on Computer Science Education (SIGCSE), March 2013.

[19] J. Sorva. Visual Program Simulation in Introductory Programming Education. Ph.D. dissertation, Aalto University, 2012.

[20] D. Pritchard and T. Vasiga. CS Circles: An In-Browser Python Course for Beginners. In Proceedings of the ACM technical symposium on Computer Science Education, SIGCSE '13, 2013.

[21] B. Miller and D. Ranum. Beyond PDF and ePub: toward an interactive textbook. In Proceedings of the 17th ACM annual conference on Innovation and technology in computer science education, ITiCSE '12, pages 150–155, New York, NY, USA, 2012. ACM.

[22] A. D. Edgcomb, F. Vahid, R. Lysecky, A. Knoesen, R. Amirtharajah, M. L. Dorf, Student performance improvement using interactive textbooks: A three-university cross-semester analysis, 122nd ASEE Annual Conference and Exposition: Making Value for Society.

[23] C. Meyers and T.B. Jones, Promoting Active Learning: Strategies for the College Classroom. 1993: Jossey Bass Pub; San Francisco.