An Investigation On Student Perceptions of Self-Regulated Learning In An Introductory Computer Programming Course.

Pratibha Menon
menon@calu.edu

Department of Computer Science,
Information Systems and Electrical Technology
California University of Pennsylvania
California, PA, 15419

Abstract

Learning how to become a self-regulated learner could benefit students in introductory level, undergraduate courses, such as computer programming. This study explores the role of various learning activities that can be used in an introductory programming course to develop the skills required to support self-regulated learning. The design of the learning activities is guided by a teaching and learning model, and a model for self-regulated learning. Students' perception of the value of various types of learning activities is compared with their perceived confidence in applying self-regulated learning skills for independent mastery, problem-solving, correcting errors, and experimenting with programs.

Keywords: Computer-programming, Self-Regulated-Learning, independent-mastery, problem-solving, teaching, learning.

1. INTRODUCTION

Introductory computing courses are generally regarded as difficult, and often see significant number of drop outs that leads to attrition (Kinnunen & Malmi, 2006). According to (Beaubouef & Mason 2005), most attrition occurs during freshman and sophomore years. Studies have also shown that students often do not acquire an adequate level of practice as they complete their introductory computing courses (Lister et al., 2004). One approach to increasing success rates in undergraduate computer programming courses is by teaching students how to become more effective self-regulated learners who will apply deliberate practice to improve their programming skills.

Self-regulated learning is an active process where the learner attains the desire to be independent in their learning. They set learning goals, monitor their goals, regulate their cognition, motivation, and behavior towards achieving their set goals (Pintrich, 2004). Self-regulated learners take greater initiative in their learning process and persevere by constantly adapting to the tasks at hand (Zimmerman, 2002).

Becoming a self-regulated learner could especially benefit students in challenging undergraduate courses, such as computer programming. The majority of learning in a computer programming course takes place outside the classroom, as it involves hands-on practice in writing, compiling, and testing computer programs. However, many college students are not effective self-regulated learners (Bembenutty, 2008). Freshmen students often rely on the support of their teachers during secondary schooling to direct their learning processes. Therefore, many freshmen students find it challenging to engage in self-directed learning that requires repetitions of cycles of planning-practice-and-reflection.

This study explores the value of various learning activities that can be used to teach cognitive skills required to support self-regulated learning. The context of this study is an undergraduate level, introductory programming course. This study was conducted in a class of 22 students, at a public university. The study considers various teaching and learning approaches used to model and instruct the cognitive strategies required to apply self-regulated learning in a computer programming class. This study attempts to find the perceived self-efficacy of students to learn programming, after they were exposed to the learning activities designed to promote SRL. The perceived value of each of the learning activities of the perceived self-efficacy of students is analyzed, in an effort to guide future design of teaching and learning activities.

2. RELATED WORK

This study assumes that learning the practice of computer programming takes place as a cyclical exchange of knowledge and information between the learner and an external learning environment. Besides the interaction of the learner with external agents, a learner is also assumed to go through an internal process that regulates the thoughts and actions within the mind of the learner. A Self-Regulated-Learning (SRL) model is used to identify various steps in the thought process of a self-regulated learner.

2.1 The teaching-learning model

For this study, the interaction of the learner with the learning environment is assumed to take place in two ways; 1) between the learner and the teacher, and 2) between the learner and an external learning tools such as an Integrated Development Environment (IDE). These interactions may be termed as the Teacher-Practice cycle and the Teacher-Modeling cycle (Laullilard, 2012). The Teacher-Practice cycles involve cycles of interaction in which the teacher elaborates and displays the ideal way to practice a skill and provides feedback to the learner questions. The teacher-modeling cycle considers the interaction between the learner and the learning tool, which in this study is the IDE. Teacher-Modeling cycles influence the learner's abilities to engage in independent and deliberate practice to improve programming and problem-solving skills. The IDE provides immediate feedback to students and provides opportunities for students to engage in repeated practice and self-regulated learning. Although there might be several relevant learning interaction between the learners, or between a learner-tutor and learner-Internet these are beyond the scope of this paper In a programming course, the Teacher-Practice cycle typically consists of code-demonstrations that are used to discuss coding and problem-solving practices. The Teacher-Modeling cycle is enabled through suitable practice problems that require the use of an IDE to implement solutions. A teacher may provide additional feedback and support to help students understand and to appropriate actions based on the feedback provided by the IDE.

2.2 The Self-Regulated Learning Model

Self-Regulated-Learning (SRL) is a research area under which a considerable number of variables that influence learning such as, self-efficacy, volition, and cognitive strategies are studied within a comprehensive and holistic approach. A meta-analytic study of SRL identifies various models that researchers can utilize those that better suit their research goals and focus (Panadero, 2017). This study draws from previous studies on SRL that posits that Self-regulated learning can be taught (Pintrich & Zusho, 2002). SRL strategies can be transferred to students through instructions that are specific to the learning context (Perels, Dignath, & Schmitz, 2009). These studies show that providing direct instructions on specific strategies, and the use of an adequate learning environment can enhance students' self-regulated learning. The instruction on the use of a specific self-regulation strategy should explain how to apply the strategy, when to use it, and why it should be done.

The model of SRL that is used in this study is derived from the work of Zimmerman (Zimmerman & Moylan, 2009). Zimmerman's SRL model is organized into three phases: forethought, performance, and self-reflection. In the forethought phase, the students analyze the task, set goals, and plan how to reach them.

In the performance phase, students execute the task, monitor their progress, and use self-control strategies to keep themselves cognitively engaged and motivated to finish the task. Finally, in the self-reflection phase, students assess and try to understand the factors that might have impacted their success or failure. The self-reflection phase generates reactions that can positively or negatively influence how the students approach the task in later performances.

The SRL model includes the regulation of cognition, motivation, and emotions. Zimmerman's cyclical phase model has been tested in a series of studies. Studies that compare experts and non-experts in sports show that

experts performed more SRL actions (Cleary & Zimmerman, 2001) (Cleary et.al, 2006).

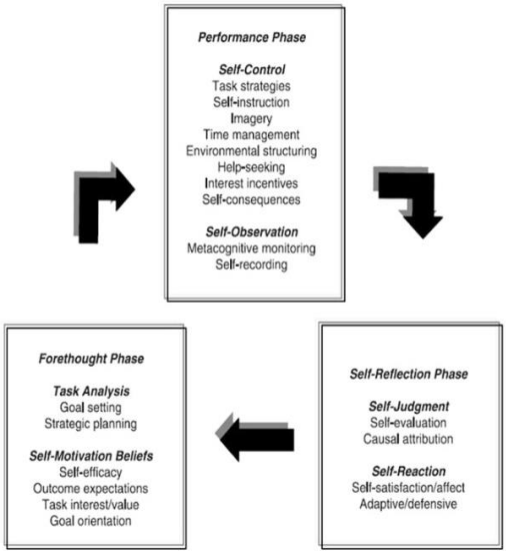

Figure 1 . Zimmerman's Self-Regulated-Model (Zimmerman & Moylan, 2009)

Zimmerman's three-phase SRL model could be applied to model the learning process in a computer programming course. In a programming course, students need to analyze the requirements of the task and constantly monitor their code to find errors (syntactical, logical, and runtime errors) before arriving on an acceptable programming solution. After completing a task, it would help the students to and reflect on their coding habits and practices so that they can improve their performance next time. By providing students with suitable instruction during the Teacher-Practice cycles, the teacher can model different ways by which students may monitor their practices. Students could apply these learning strategies to take control of their learning during the Teacher-Modeling cycles.

Previous studies have examined the role of self-regulation within the educational context of computer programming (Bergin, et. al, 2002) (Kumar et. al, 2005) (Chen, 2020) (Ramirez, et. al, 2018). The focus of these studies has been to evaluate the impact of self-regulated-learning strategies on the coding performance of students. Another study by Castellanos, et. al, uses the source code produced by students to study student motivation, performance, use of learning strategies (Castellanos, 2017). On the contrary to the two aforementioned studies, the study described in this paper evaluates the perceived value of teaching and learning activities and the perceived ability of students to acquire the cognitive skills required for self-regulated learning in a programming course by learning appropriate cognitive and metacognitive strategies through source code development.

Extensive recent work on SRL exists in the area of building online, adaptive learning systems with open-learner-models (OLMs) that allows learners to visualize and inspect their progress during the learning process. It has been pointed out that OLM can support metacognition and self-regulation (Bull & Kay, 2013). Moreover, researchers have incorporated the use of OLM in all phases of self-regulation, i.e. preparation, performance and appraisal, and in the areas of cognitive, metacognitive, motivational, and emotional support (Hooshyar et.al, 2020). For example, OLM has been used to improve self-assessment accuracy incorporating dialog-based support (Suleman et.al, 2016), and to improve engagement in a programming course (Hossieni et.al, 2020). All these studies were performed in the context of full-online learning that doesn't include any direct intervention by a teacher during the learning process. The study described in this paper models a typical freshman-level, under-graduate class room scenario, in which the teacher still plays a central role in mediating the self-regulation strategies of students. Therefore, the focus of this paper is on a teaching-learning model that includes the central role of a teacher in designing and supporting the learning process by adapting to the needs of the learners.

3. THE DESIGN APPROACH

The instructional design that is evaluated in this study incorporates teaching strategies for the Teacher-Practice cycle, and suitable learning activities for the Teaching-Modeling cycle. The teaching methods are chosen such that they incorporate the three phases of Zimmerman's SRL model.

3.1 Designing the Instructional Activities

The teacher-practice learning cycle consists of activities through which the instructor, who is also an expert programmer, models the programming practices. Table 1 shows the instructional activities in the Teacher-Practice cycle. Code-demonstrations (code-demos) provide an elaborate explanation of the programming process through task analysis, code development, execution, and testing. The sample code used for the code-demo contains extensive documentation and comments that students can refer to later on.

The forethought/planning phase of the code-demo typically includes a detailed explanation and analysis of the problem statement to identify the functional and data requirements. These planning activities are written down as part of the code documentation, in the comments section of the code. The instructor may use real-world examples to show the value of the problem. The inputs and the expected outputs are identified and a test plan is created.

The performance phase of the code-demo typically involves the instructor elaborating on the systematic thought process required to write the program sequences. Some of these thoughts are written as comments next to the code statements. Techniques like tracing the variables, or printing out the values of the variables are used to help students test and incrementally build their code.

	Instructional activities - Teacher-Practce Learning Cycle		
	Forethought	**Performance**	**Self-Reflection**
Code Demos	Problem Analysis, Solution planning, Reviewing Test Plans	Choice of constructs, Identifying right sequence, Tracing variables, Running Tests	Evaluating Style & Practices and Errors
Q&A Sessions	Task planning , Goal Setting for the class	Discussions on Identifying and correcting errors; adopting good practices	Choosing practice materials to strengthen practice

Table 1. Instruction Activities – Teacher-Practice Learning Cycle

The self-reflection phase of each code-demo is used to analyze various options for accomplishing the same outputs. Good coding practices, relevant to the problem, are also discussed. Students are encouraged to reflect upon the challenges they encountered while solving the problem and ways in which they can improve their problem solving and programming skills.

Integral to the Teacher-Practice learning cycle are the Q&A sessions. The Q&A sessions are conducted after each learning activity session described in Table 2. During the Q&A sessions, besides answering questions to clear any misconceptions, or problem solving difficulties that students would have experienced while completing a learning activity. The instructor may also discuss the assignments and some of the common errors and misconceptions that would have appeared in student submissions.

3.2 Designing Practice Exercises

The Teacher-Practice cycle is followed by the Teacher-Modeling cycle, during which students are expected to apply the application development practices discussed by the teacher. Students are assigned several different types of practice exercises as class activities.

The Do-It-Yourself (DIY) exercises, which are not exactly the same, but very similar to the problems explained during the code-demos, let students try out the sample-code written by the instructor. By observing the sample code, students can emulate the practices of the instructor and apply all three phases of SRL to document and write the code by themselves using an IDE. DIY activities require students to read the instructor's detailed comments and check their understanding before they begin to write the code by themselves. The code samples of the DIY activities were free of errors and contains the coding best-practices. The DIY activities also advise students to incrementally build their code rather than just copy the entire code all at once. A sample DIY activity is shown in Appendix B.

Activity name	**Practice Exercises - Teacher-Modeling Learnng Cycle**
DIY	Try out every code-demo independently, following the instructor's comments/explanation.
Test-Tube	Test a given code by varying the inputs,or by making suggested changes to obtain a given output
Messed Up Code	Analyze an errored-code
Hack the Code	Experiment with a given code to produce a set of outputs (including errors)

Table 2. Type of Practice Exercises – Teacher-Modeling Cycle

It has been commonly noticed by several instructors that novice programmers are often challenged by the programming errors that they generate while learning to write programs. Many students require help to understand the types of errors and how they can learn from their mistakes to improve their programming skills.

To help students gain practice and become comfortable with detecting and correcting logical and syntax errors, activities called Hack-the-code and Messed-up-code have been developed by the instructor. The Messed-up code contains one or more errors that students need to identify and correct. Hack-the-code is an activity in which students are given a piece of written code whose logic they need to alter to obtain the required set of outputs. The Messed-up code and Hack-the-code activities intend to encourage students to feel comfortable in experimenting with their code. Another activity that encourages experimentation

is Test-Tube activity. This activity requires students to develop and execute a test-plan for a given code and in most cases, also requires them to trace the variables. All these activities are designed to teach cognitive and meta-cognitive strategies that improve coding practice. Samples of Hack-the-code, Test-Tube and Messed-up code activities are shown in Appendix B.

The goal of the learning activities is to let students work on the problems by themselves and learn how to ask for help from their peers and instructor. Students are encouraged to apply the three phases: task analysis, performance monitoring, and self-reflection for every task they perform. The Q&A sessions were specially geared towards addressing the problems students faced while working on the activities. The class activities were expected to be completed during the class time and students received class participation points for attempting, and not necessarily completing these activities.

4. THE STUDY

The main intent of this study is to collect, and to analyze, the perceptions of students on their ability to apply the skills required for self-regulated learning in an introductory programming course. This study was conducted in an undergraduate computer programming course that teaches introductory programming using Java. An initial survey was conducted at the beginning of the course to gauge the concerns of students in regards to learning a programming course. This survey was also used to measure the prior knowledge of programming and the perceived self-efficacy to learn programming at the beginning of the course. A final, end-of-the-course survey was conducted to study the student perceptions on the usefulness of various learning activities that formed the instruction of the course, and the perceived self-efficacy of students to learn computer programming at the end of the course.

Both the initial and final surveys questions are shown in Appendix A. Both the surveys used a 5-point Likert scale to score student responses. Twenty students attempted the initial survey and only nineteen students attempted the final survey. This discrepancy in the number of students was accepted because the initial and final results weren't matched, compared or correlated. Students weren't individually identified in the survey and there was no matching of data collected during the initial and final surveys. These surveys were anonymously administered to students. The initial survey results were used only to understand the prior experience and perceived self-efficacy of students prior to attending the course. The final survey was intended to study the student perceptions at the end of the course, whose instruction was primarily made up on programming activities that was designed to promote self-regulated learning skills. The final survey responses are not used to show how much the student perceptions changed as a result of the instruction in the course. Rather, this study looks into the perceived usefulness of different types of practice activities that constituted the instruction in the programming course.

Students were required to attempt all the assigned learning activities assigned in a set sequence. The scope of this study is limited to evaluating the cognitive learning strategies required to develop self-regulated learning in the programming course. It is assumed that practicing these cognitive skills would give enough learning experience to help students regulate their attitudes and behavior towards learning how to write programs. The perceived effectiveness of the learning strategies could be impacted by motivation, beliefs, and emotional factors that were not directly addressed through the instruction.

5. RESULTS

The results from the initial and final surveys are discussed in this section. The pre-course-implementation survey measures the students' concerns, prior programming skills, student beliefs, and learning preferences. The post-course-implementation survey measures the perceptions of students towards various learning activities designed for the Teacher-Practice and Teacher-Modeling cycles. The post-survey also measures the perceived confidence in applying various self-regulated learning skills, as a result of their experience with the learning activities.

5.1 Students' perception of their self-efficacy

Table 3 shows the results of an initial survey conducted during the first week of the course. It is seen that students were less concerned about how much they can master the contents of this course than they were about having the right skills and abilities to learn to program. The survey was administered to students after the course syllabus was discussed by the instructor.

Table 3 also indicates the self-reported prior experience with computer programming. Since data that is collected using a 5-point Likert scale

is ordinal, a Spearman-rank correlation method is chosen to investigate the correlation between the degree of prior exposure to computer programming, and the learning concerns reported by students.

	Very Much Disagree	Somewhat Disagree	Neutral	Somewhat Agree	Very Much Agree
I am concerned about how much I can master the subject matter	2	3	2	7	6
I am concerned if I have the right skills to learn programming	1	6	6	3	4
I have some prior knowledge of programming	8	3	2	3	4

Table 3. Survey response distributions on the perceived self-efficacy and programming knowledge - prior to the course.

It was found that the degree of prior exposure to programming was negatively correlated (, with a moderately significant correlation coefficient, rho of -0.6, p = 0.005) with the students' concerns about having the right skills to learn to program. The correlation between prior exposure to programming and concerns about learning the subject matter was weak (rho = -0.3, p=0.005). These results show that students with lesser prior exposure to programming were more concerned about their preparedness for acquiring programming skills than they were on their readiness to learn the knowledge contained in the subject matter. This result pointed to the possibility that an instructional strategy that included more activities to build programming skill, might be valuable to develop the perceived self-efficacy of students in their ability to develop programming skills.

5.2 Student Perceptions of independent/ self-directed study –prior to the course

The initial survey conducted at the beginning of the course, indicated responses on some of the prior beliefs and preferences that students bring to the course. As shown in Table 4, a large number of students do not believe that they can master programming through independent/ self-directed study, although many did indicate that they tend to reflect on the problems when they feel stuck in their assignments.

	Very Much Disagree	Somewhat Disagree	Neutral	Somewhat Agree	Very Much Agree
When I am stuck with problem(s) in assignments, I tend to reflect on 'why exactly I am stuck'	**2**	**4**	**3**	**9**	**2**
I generally ask for help so that I can quickly finish the assignments	**0**	**8**	**7**	**2**	**3**
I believe that one can master programming only by working independently on hands-on learning activities.	**3**	**7**	**2**	**5**	**3**
I think it is important to know not just what to learn, but also how to learn that subject.	**0**	**0**	**0**	**13**	**7**

Table 4. Survey response distributions on the perceived ability for self-directed/independent study - prior to the course

Results also indicated that only a few students would like to get help just so that they can finish their tasks quickly. It appeared from the results that almost all the students were open to learning the skills that are necessary to master a subject through self-directed study.

5.3 Perceptions of the effectiveness of learning activities after the course

The final survey results showed the perceived value of various learning activities that became a regular part of instruction throughout the semester. Table 5 shows that a majority of students agreed that practicing and participating in these learning activities were valuable in acquiring the programming skills that they were expected to learn from the course.

All the activities, except for the Q&A sessions, required students to apply their knowledge and skills to identify the problem, plan the solution,

write the code, correct errors, and test the code –all by themselves. These activities provided different ways for students to apply one or more SRL skills related to learning how to develop programming solutions. The Q&A sessions was the time when students obtained help and feedback from the instructor.

	Very Much Disagree	Disagree	Neutral	Agree	Very Much Agree
Q&A sessions	0	0	1	11	7
Messed Up Code	0	0	1	12	6
Hack the Code	0	0	1	11	7
Test Tube	0	0	1	7	11
DIY	0	0	4	13	2

Table 5. Student response distribution on the effectiveness of learning methods in developing programming skills – final survey results

Survey results on students' perceptions, depicted in Table 6, showed that 18 out of 19 respondents agree or strongly agree that they feel comfortable to experiment with their code. All the respondents also report that they feel confident in their ability to correct programming errors. Out of the 19 respondents, 16 (, which is 85% of respondents) feel that learning how to program has improved their problem-solving skills. However nearly 9 out of 19 (, which is 48% of) respondents don't believe that they can master programming only by working independently on hands-on activities. The popularity of the Q&A sessions, as shown in Table 5 shows that students find the help obtained through the Q&A , as valuable as the self-directed study.

5.4 Correlation studies

A Spearman-Rho correlation analysis was used to study the correlation between various factors indicating the perceptions of self-efficacy and self-directed learning (listed in Table 6,) and the perceived value of various instructional methods (listed in Table 5). For the sample of 19 respondents, it was observed that there existed a significant correlation (, rho = 0.65, p=0.005) between the belief of students on their ability to independently master programming and the perceived value of doing many DIY activities.

	Very Much Disagree	Disagree	Neutral	Agree	Very Much Agree
I feel that learning how to program has improved my problem solving skills	0	0	3	7	9
I feel confident to experiment with my programs	0	0	1	8	10
I feel confident that I can correct programming errors	0	0	0	9	10
I believe that one can master programming only by working on independently on hands-on activities	1	2	6	7	3

Table 6. Student response distribution on the perceptions of the self-efficacy and self-directed learning – final survey results

No significant correlation was found to ascertain that the perceived values of Test-tube or Hack-the-code are associated with any of the factors that indicate the perceived abilities (, as listed in Table 6) to learn how to program. However, a moderately significant correlation (rho = 0.57, p=0.005) was detected between the value of the Messed-up-code activity and the ability to master programming by independent learning. A correlation (rho = 0.6, p= 0.005) was found between the value of the Q&A sessions and the perception that learning to program has improved their problem-solving skills.

The perceived confidence in problem-solving skills was significantly correlated with confidence in correcting errors (rho = 0.62, p=0.005), and with the confidence in learning from mistakes (rho = 0.6, p=0.005). However, the correlation between confidence in problem solving and belief in independent mastery, wasn't statistically significant. Based on the results, even if a student is confident in problem-solving, the student did not necessarily believe that independent mastery of programming is possible. The confidence in problem solving is also not significantly correlated to the perceived value of the hands-on learning activities, for the given student sample.

5.5 The Instructor's reflection on the results

Both the Q&A session and the DIY activities involved the instructor's participation and support to a much greater extent than did the Test-tube, Hack-the-code, or the Messed-up-code activities. The DIY activities did not require students to troubleshoot or to apply problem analysis, as much as the other activities required them to do. The DIY activities resembled mini-projects, while the other learning activities were mostly like skill-builder activities. The value of completing the DIY programs, by 'walking in the shoes' of the instructor seems to correlate more with the beliefs that students can master programming through independent practice.

By reflecting upon the classroom experience, it was observed that students did not require much help from the instructor to complete the DIY activities. The code documentation and worksheets were very elaborate and there were plenty of comments, a readymade test plan, and test cases. Students could complete the DIY program perfectly with very little help once they start the task. On the other hand, the other activities came with nothing more than a problem statement and the expected output(s). To complete the Test-tube, Hack-the-code, and Messed-up-code activities, students required help on various aspects of problem-solving such as, understanding the task, identifying the required variables and logic, identifying the errors, etc.

From an instructor's perspective, seeking help is an important skill required for self-regulated learning. However, the belief that one can master programming through independent learning wasn't strongly correlated to the perceived value of the Test-tube and Hack-the-code activities, for which students needed more help. A student who considers Test-tube and Hack-the-code as valuable to their learning , is still not likely to say that they believe they can master programming independently, possibly because they needed more help and support to complete the tasks. Compared to the Test-tube and Hack-the-code activities, the Messed-up-code did not require students to alter the inputs. Therefore, from an SRL standpoint, Test-tube and Hack-the-code involved a lengthier thought process than what was required for the Messed-up-code activity.

In addition to needing more help with the Test-tube, Messed-up-code, and Hack-the-code, students tended to make more mistakes, even though they would eventually figure out a way to correct the mistakes. From an instructor's perspective making and correcting mistakes indicates self-regulated learning. However, if students perceive mistakes negatively, they are less likely to register these activities as contributing to their confidence to learn. This is possibly the reason why despite the perceived value of the Test-tube, Messed-up-code, and Hack-the-code, they were not correlated to the confidence for independent mastery. These learning activities were not high-stakes graded activities and students received participation points just for attempting them. Perhaps, the instructor needs to find ways to give incentives to students to self-reflect on how the mistakes have improved their performance.

The Q&A sessions were designed to encourage students to ask questions. Most of the Q&A sessions were associated with the weekly assignments and the class activities such as the Test-tube, Hack-the-code, and Messed-up-code. A significant correlation between confidence in problem-solving skills and the value of Q&A indicates that students are likely to view help and support as factors that improve their problem-solving.

6. CONCLUSIONS

This study investigates the student perceptions of the role of teacher-practice activities and teacher-modeling activities in an introductory computer programming class. The majority of the students agree that the hands-on learning activities had greatly helped them to acquire the programming skills, even though more than half of the students reported that they were not confident in their abilities to master programming independently. Emulating the instructor's coding process through the DIY activities is what the students found as most valuable in mastering their programming skills independently; and the Q&A sessions were strongly perceived and correlated with confidence in problem-solving skills. Future iterations of the course could consider tweaking the self-directed learning activities so that students can see the value of making mistakes and getting help, in their ability to master programming independently.

This study does not objectively evaluate how realistic are the students in reporting their perceived confidence in applying SRL. Due to the limitations of the approved research protocols, no subject research could be conducted to identify students and relate the survey responses to their performance in class activities. However, irrespective of their performance, or even their actual growth in self-regulated learning, how students feel about their self-regulated learning skills matter in their future decisions to engage in

programming courses. Future studies can look into learning strategies that could help students regulate their behavior and motivation as they encounter greater challenges in their learning process.

7. ACKNOWLEDGEMENTS

The author would like to thank and acknowledge the PASSHE-FPDC grant for funding the instructor during summer 2019. The grant funding helped the author to acquire the required professional development on Self-Regulated Learning and Design Thinking that was applied in this study.

8. REFERENCES

Beaubouef, T., & Mason, J. (2005). Why the high attrition rate for computer science students: some thoughts and observations. *ACM SIGCSE Bulletin, 37*(2), 103-106.

Bembenutty, H. (2008). The teacher of teachers talks about learning to learn: An interview with Wilbert (Bill) J. McKeachie. Teaching of Psychology, 35, 363–372.

Bergin, S., Reilly, R., & Traynor, D. (2005) "Examining the role of self-regulated learning on introductory programming performance", *Proceedings of the first international workshop on Computing education research*, pp. 81-86, 2005.

Bull, S & Kay, J. (2013) "Open learner models as drivers for metacognitive processes," in International Handbook of Metacognition and Learning Technologies. Springer, pp. 349–365.

Castellanos, F. H. Restrepo-Calle, F., González. A, & Echeverry, J. R. (2017) "Understanding the relationships between self-regulated learning and students source code in a computer programming course," *2017 IEEE Frontiers in Education Conference (FIE)*, Indianapolis, IN, 2017, pp. 1-9,.

Chen, C. S., (2002) "Self-regulated learning strategies and achievement in an introduction to information systems course", *Information technology learning and performance journal*, vol. 20, no. no. 1, pp. 11.

Cleary, T., Zimmerman, B. J., & Keating, T. (2006). Training physical education students to self-regulate during basketball free throw practice. *Res. Q. Exerc. Sport* 77, 251–262.

Cleary, T., & Zimmerman, B. J. (2012). "A cyclical self-regulatory account of student engagement: theoretical foundations and applications," in *Handbook of Research on Student Engagement*, eds S. L. Christenson and W. Reschley (New York, NY: Springer Science), 237–257.

Hooshyar, D. M., Pedaste, K., Saks, ¨A., Leijen, E. Bardone, & Wang, M. (2020) "Open learner models in supporting self-regulated learning in higher education: A systematic literature review, "Computers & Education, p.103878

Hosseini, R., Akhuseyinoglu, K., Brusilovsky, P., Malmi, L., Pollari-Malmi, K., Schunn, C., and Sirkiä, T. (2020) Improving Engagement in Program Construction Examples for Learning Python Programming. *International Journal of Artificial Intelligence in Education*.

Kinnunen, P., & Malmi, L. (2006). *Why students drop out CS1 course?* Paper presented at the Proceedings of the second international workshop on Computing education research.

Kumar, V., Winne, P., Hadwin, A., Nesbit, J. et al., (2005) "Effects of self-regulated learning in programming", *Advanced Learning Technologies. ICALT 2005. Fifth IEEE International Conference on*, pp. 383-387.

Laurillard, D. (2012) *Teaching as a Design Science*. (New York: Routledge).

Panadero, E & Alonso-Tapia, J. (2014). How do students self-regulate? Review of Zimmerman's cyclical model of self-regulated learning. Anales de Psicología. 30. 450-462.

Panadero E. (2017). A Review of Self-regulated Learning: Six Models and Four Directions for Research. *Frontiers in psychology*, *8*, 422. https://doi.org/10.3389/fpsyg.2017.00422.

Perels, F., Dignath, C., & Schmitz, B. (2009). Is it possible to improve mathematical achievement by means of self-regulation strategies? Evaluation of an intervention in regular math classes. *European Journal of Psychology of Education, 24*(1), 17.

Pintrich, P. (2004). A conceptual framework for assessing motivation and self–regulated learning in college students. Educational Psychology Review, 16, 385–407.

Pintrich, P. R. (2000). The role of goal orientation in self-regulated learning. In M. Boekaerts, P.R. Pintrich, & M. Zeidner (Eds.), *Handbook of self-regulation (pp. 451–502)* (pp. 451–502). San Diego, CA: Academic Press.

Pintrich, P. R., & Zusho, A. (2002). *The development of academic self-regulation: The role of cognitive and motivational factors.* In

A. Wigfield & J. S. Eccles (Eds.), *A Vol. in the educational psychology series. Development of achievement motivation* (p. 249–284). Academic Press.

Ramírez, J. J. E , Rosales-Castro, L. F. , Restrepo-Calle & González, F. A. (2018) "Self-Regulated Learning in a Computer Programming Course," in IEEE Revista Iberoamericana de Tecnologias del Aprendizaje, vol. 13, no. 2, pp. 75-83.

Suleman, R. M. , Mizoguchi. R & Ikeda. M, (2016) "A new perspective of negotiation-based dialog to enhance metacognitive skills in the context of open learner models, "International Journal of Artificial Intelligence in Education, vol. 26, no. 4, pp. 1069–1115, publisher: Springer.

Zimmerman, B. J., & Moylan, A. R. (2009). Self-regulation: Where metacognition and motivation intersect. In *Handbook of metacognition in education* (pp. 311-328). Routledge.

Appendix A

Initial survey - conducted prior to the course					
	Very Much Disagree- 0	**Disagree- 1**	**Neutral- 2**	**Agree - 3**	**Very Much Agree-4**
I am concerned about how much I can master the subject matter in this course					
I am concerned if I have the right skills to learn programming					
I have some prior knowledge of programming					
I believe that one can master programming (or any subject) only by working independently on hands on activities.					
I think it is important to know not just what to learn, but also how to learn that subject.					
When I am stuck with problem(s) in assignments, I tend to reflect on 'why I am stuck'.					
I generally ask for help so that I can quickly finish the assignments					

Final Survey - conducted at the end of the course					
Please answer these questions based on your learning experience in the CIS 120 course	**Very Much Disagree- 0**	**Disagree- 1**	**Neutral- 2**	**Agree - 3**	**Very Much Agree-4**
The Q&A session is a valuable learning method for this course					
Test-Tube: Experiementing with code is a valuable learning method for this course					
DIY : Trying out the code-demos using Eclipse is a valuable learning method for this course					
Messed-up-code: Analyzing and fixing an errored code is a vauable learning method for this course					
Hack-the-code: Experimenting with the code to alter the outputs helped me learn better					
I believe that one can master programming by working only independently on hands on activities.					
I feel that learning how to program has improved my problem solving skills					
I feel confident to experiment with my programs					
I feel confident that I can correct programming errors					

Appendix B

1. **A Sample DIY problem:**

Shopping Cart – Create a file called ShoppingCart.java

Please refer to the code demo called **VariableDataEntry.java** prior to attempting this problem. This problem shows you how to:

- obtain data from the user, scan this data and save it in an appropriate variable.
- perform arithmetic using the numeric data types,
- print a message displaying values of all the variables.

In this program you will capture data of an item for a ShoppingCart application. Your program may need to know the following properties: customer_name, item_name, item price, sales tax rate, item quantity, calculated total price of all items in the cart

A ShoppingCart may need the following behaviors:

- Obtain the following data from the user for for a single item: customer_name, item_price, sales_tax_rate, item_quantity. Scan these values and store them in variables of appropriate data type.
- Calculate the total price of all items in the cart
- Print a message listing all the item variables with its total calculated price (that includes the sales_tax factored in).

2. **A Sample Hack-the-Code activity**:

Refer to the code called AgeCheckerCase2.java.

```
import java.util.Scanner;
public class AgeCheckerCase2 {
    public static void main(String[] args){

        //Declare the input variable
        int age = 0;

        //Declare output variable
        double ticketPrice = 0.0;

        //Declare other variables
        double salesTax = 0.05;

        //Decision structure
        if(age < 12) {
            salesTax = 0.2;
        }

        ticketPrice = ticketPrice*(1+salesTax);

    }
}
```

Hack this code so that your decision structure calculates the ticketPrice based on the following rule: For an age that is less than 12 , give a 20% discount on ticketPrice , but for an age greater than 65, give just 10% discount on the ticketPrice for all other age groups between and including 12 and 65, give just 2% discount on ticketPrice.

ISSN: 2473-4901
v6 n5358

3. A Sample Test-Tube activity

```
public class TracingWhileLoops_3 {
    public static void main(String[] args){

        int i = 0;
        int result = 0;
        int gate = 5;
        int n = 1;

        while(i<gate){

            if(i/4 == 0){
                result = result + 1;
            }
         i = i + n;
        }
    }
}
```

1. Determine the value of result, i/4 and (i<gate) for each iteration of the while loop and complete the table shown below

gate = 5	n =2	i	result	i/4	i<gate
5	2	0	0		
5	2				
5	2				
5	2				
5	2				
5	2				

2. Determine the value of result, i/4 and (i<gate) for each iteration of the while loop and complete the table shown below for a gate = 10 and n = 3. Add more rows if needed.

gate = 10	n =3	i	result	i/4	i<gate
5	2	2	0		
5	2				
5	2				
5	2				
5	2				
5	2				

A Sample Messed-up Code Activity

Problem : Use decision structures to check if a variable **userLetter** is a vowel in the English alphabet. Assume the value of userLetter is already obtained from the user and set to an appropriate data type in each of the following responses. Correct the errors each of the following responses that assumes a given data type for userLetter,

Response 1: userLetter is a String.

```
if (userLetter.equalsIgnoreCase "a"){
System.out.println("Letter is a vowel");
}

if (userLetter.equalsIgnoreCase "e"){
System.out.println("Letter is a vowel");
}

if (userLetter.equalsIgnoreCase "i"){
System.out.println("Letter is a vowel");
}

if (userLetter.equalsIgnoreCase "o"){
System.out.println("Letter is a vowel");
}

if (userLetter.equalsIgnoreCase "u"){
System.out.println("Letter is a vowel");
}

else{
System.out.println("Letter is not a vowel");
}
```

Response 2: userLetter is a char

```
if(user == a){
            System.out.println("Its a Vowel ");
        }
else if (user == e){
            System.out.println("Its a Vowel ");
        }
else if (user == i){
            System.out.println("Its a Vowel ");
        }
else if (user == o){
            System.out.println("Its a Vowel ");
        }
else if (user == u){
            System.out.println("Its a Vowel ");
        }
else {
           System.out.println("Not a vowel ");
        }
```

Response 3: userLetter is a String and you need to use a || in your if condition

```
if else(letter.equalsIgnoreCase("A||E||I||O||U")){
        System.out.println("you got a vowel");
}
```

Response 4: userLetter is a char and you need to use a || in your if condition

```
if (userLetter = a || e || I || o || u) {
System.out.println("This letter is a vowel.");
else if () {
System.out.println("This letter is not a vowel.");}
```