C.F. Reilly and J.A. Swanson. A case study in constructivist pedagogy in a computer organization course. To appear in FIE 2019: https://fie2019.org/

© 2019 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collective works, for resale or redistribution to servers or lists, or reuse of any copyrighted component of this work in other works.

A Case Study in Constructivist Pedagogy in a Computer Organization Course

Christine F. Reilly
Computer Science Department
Skidmore College
Saratoga Springs, New York, USA
creilly@skidmore.edu

Joan Ann Swanson
Education Studies Department
Skidmore College
Saratoga Springs, New York, USA
jswanson@skidmore.edu

***Abstract*—This Full Paper in the Research to Practice category presents an analysis of a constructivist pedagogical approach for the semester-long programming project in an Introduction to Computer Organization course at a liberal arts college in the Northeast United States. At this college, the Computer Science (CS) major has a relatively small number of required courses. In order to ensure that CS majors experience an appropriate breadth and depth in the field and because this course is required for all CS majors, the programming project was designed to meet and reinforce many of the learning outcomes for the major. After the end of the semester, a survey was administered to the students in order to obtain feedback about the usefulness of the programming project in the context of the course, and in the context of the CS major. The methods discussed in this paper are directly useful to other departments that have a small number of required courses. Additionally, these methods could be utilized by larger departments that wish to provide a reinforcement of concepts across multiple courses.**

***Index Terms*—Computer Organization, Constructivist Pedagogy, Course Project**

I. Introduction

This paper describes a programming project that was developed for an Introduction to Computer Organization course at a small liberal arts college in the Northeast United States. A constructivist pedagogical approach was utilized in the design of this programming project. Students were provided scaffolded opportunities to get hands-on experience with the course topics, to see how a large project is broken into smaller components, and to practice Java programming.

The programming project described in this paper is part of a redesign of the Computer Organization course at this college. The department is in a period of transition with the recent addition of two faculty members who focus on computer systems (including one of the authors of this paper), and ongoing discussions about revisions to the entire curriculum for the major.

Computer Organization is a course that is typically included in the requirements for a computer science (CS) major [1]. The structure of the CS major at this liberal arts college provides a particular context for the Computer Organization course. The approach of the CS major at this college is to have a limited number of required courses in order to allow students to take advantage of the opportunities that a liberal arts college provides for choosing to focus their studies on their own interests. Some students focus on computer science, and take many CS elective courses in addition to the courses required for the major. Other students select a second major, multiple minors, or engage in a self-determined course of study in addition to the CS major. While this approach is appropriate for the context of this college, the CS faculty are faced with the challenge of ensuring that the required courses provide students with the depth and breadth of knowledge that is expected from a computer science major. This context influenced the design decisions that were made for the programming project described in this paper.

This paper continues in Section II with an overview of the learning theory that is applied in this project, followed by Section III that provides a review of related research about Computer Organization courses and about the use of constructivist pedagogy in computer science. Section IV provides information about the Computer Organization course at this liberal arts college and presents a detailed overview of the programming project that is the focus of this study. Section V details the survey instrument that is used for gathering feedback about the programming project from students and presents results from multiple semesters of data collection. The paper then has concluding remarks and suggestions in Section VI.

II. Learning Theory

"Success as a higher education faculty member demands an understanding of pedagogical issues and learning experiences that produce students who are both confident and competent" [2]. Such an understanding goes beyond a simple description and reaches to the core of understanding the methodologies, priorities and objectives. Constructivism is a philosophical stance that embraces an approach to learning that involves actively building knowledge and understanding. Within the field of computer science education, this active approach can take many forms that result in students building skills and confidence. One main tenant of constructivism is that learners create new knowledge through a cognitive process of linking prior knowledge and experiences. The researchers in this project theorize students can use their prior experiences as a framework which throughout the semester they build upon via projects and deep inquiry. Much of this process

occurs within a socio-cultural developmental framework [3]. This is most often considered a socially contextualized process since the learner's experiences with the world and people around them are the foundation from which they interpret and negotiate the construction of new schemas [4]. Continued scaffolded projects which build upon computer science skills serve to situate the learners in activity that utilizes an authentic context and culture where the projects increase not only their abilities [5] but their confidence and self-efficacy [6] to excel as computer science students.

The constructivist educator skillfully facilitates an environment in which learners engage in a process full of experiences and discovery. It is in this environment that they have opportunity to build upon what they already know [7]. Within computer science education, the constructivist educator is tasked with designing such a course in which activities and experiences will catapult their inquiry and quest for learning. Courses and assignments are designed with the end objective in mind, following a principle of "backwards design" [8].

In the scope of building knowledge and skills, there is a range in which a learner has developed and can accomplish without assistance. Extending from there, Vygotsky [3] purports a range called the Zone of Proximal Development in which learners can obtain greater understanding and skills when they have the assistance of a more skilled individual through a process called scaffolding. The nature of computer science instruction lends itself toward collaborative activities and projects in which scaffolding can occur. In the context of a computer science course where the foundations of Computer Organization are introduced, a pedagogical approach utilizing constructivist principles provides an ideal framework from which to teach. Scaffolded assignments, providing premade frameworks, give students a place from which they can build their knowledge and self-efficacy as programmers [6]. Bruner [9] further defines scaffolding as the tailored support provided in a learning process as needed. Reflection on the learning process is enhanced by dialogue within the collaborative environment [10]. The experiences and the cultural setting of the learning environment are important factors which provide context for the learning process. This paper describes an experience with a constructivist framework for teaching and learning that utilizes the pedagogical practices of inquiry, collaboration, problem-solving, and utilization of authentic tasks and assignments.

III. Related Work

Active learning approaches have been shown to improve learning outcomes and student retention in Computer Science [11], [12] and other STEM disciplines [13]. These constructivist approaches are rooted in the central tenet of building knowledge and understanding. Often this process involves learners in authentic real-world practice such as project-based learning experiences followed by reflection [14], [15]. Jung and Suzuki [16] provide an example of providing scaffolding in a collaborative project using worked examples and grouping, followed by assessment and feedback. Another example shows the application of social-constructivist pedagogy in a Computer Architecture course where students apply their knowledge to the construction of a simulated computer system [17].

TABLE I
Course Units

Unit	Main Topics
1: Computer Mathematics	Binary and hexadecimal numbers; Binary arithmetic; Boolean algebra; Logic circuits.
2: Assembly Language	Sequential programs; Decision making structures; Looping structures; Procedures.
3: Program Execution	Single cycle datapath; Program performance; Pipelined datapath; Data and control hazards.
4: Memory Hierarchy	Computer memory components; Motivation of memory hierarchy; Cache memory.
5: Parallel Computing	Parallel processors; Distributed computing and storage systems.

Course projects where students build a computer simulation of a computer processor are a common approach used in Computer Organization and Architecture courses. A benefit of computer-based simulation projects is that students have an experience that helps them learn how a computer works, with little or no infrastructure cost and no need for an electronics laboratory classroom [18]. This course project can also be leveraged to provide students with software engineering experience by using a high-level object-oriented language to construct the simulated processor [19].

Small colleges face an array of challenges in designing a Computer Organization course that gives enough depth but also covers a wide enough range of topics. Some colleges have chosen to add a hardware lab with an additional course credit [20] or to include a microcontroller programming project [21]. Others use creative projects that simulate the processor datapath to help students develop a concrete understanding of the datapath [22].

IV. Overview of Course and Assignments

This section provides an overview of the Computer Organization course and its role within the major, and then gives details about the assignments that compose the semester-long programming project that is the focus of this study.

A. Course Overview

The Computer Organization course at this liberal arts college is a 300-level course, with a prerequisite of the 200-level Java programming and data structures course (a CS2 course). The Computer Organization course is the only required systems or hardware course in the major. The course topics are divided into five units, as listed in Table I. The semester-long programming project provides students with the opportunity to gain experience with material from Units 1 through 4.

Additionally, this semester-long programming project further develops and reinforces the programming skills that are expected from CS majors. It is critical to include a large programming project in this Computer Organization course because the other required 300-level courses in the major

are focused on computer theory and do not include much programming. Students are required to take three elective courses as part of the major, and many students take electives that include a large programming project, but there is no formal requirement to ensure that students are gaining programming experience through the elective courses. By providing a scaffolded project in this Computer Organization course, all computer science majors will have a model for how to manage a large, long-term project. They can apply this model to the open-ended projects that they encounter in some of the elective courses and in their future careers.

B. Programming Project

The overall goal of the semester-long programming project is to create a software simulation of a simple computer. The project is divided into four programming assignments that are assigned and due as the semester progresses. The structured assignments that compose the overall project provide a model for how to divide a large problem into smaller components.

Java was chosen as the language for this programming project. This decision was based on the use of Java in the CS1 and CS2 courses at this college. By sticking with the language that students already have experience with, more class time can be used for Computer Organization concepts instead of on learning a new programming language. Additionally, this course project reinforces Java programming concepts that students learned in the previous courses and strengthens the computer science major.

For each of the programming assignments, the professor provides an overview of the assignment, starter code with documentation, and test programs. In most of the assignments, the professor defines the Java class methods that students are to complete for the assignment. Students determine how to implement the methods, and write the Java code for the implementation. Students may choose to add private methods to the classes they implement, but they may not alter the interface of the public methods that are defined in the starter code or define new public methods. The professor strongly encourages students to work in pairs, and students can choose a new partner or choose to work individually on each assignment.

When the code developed for an earlier assignment is used in a later assignment, the professor provides the solution for the earlier assignment code. The ideal situation would be for every student to use their own code from the earlier assignment, but some students may not correctly finish the earlier assignment. The professor chose to provide the earlier assignment solution as a Java Archive (JAR) file that contains the Java class files for the earlier code. This means that students are not able to view the exact code solution for the earlier assignment, and enables reuse of the assignment in subsequent semesters without concerns about whether the exact solution code is in circulation. The professor is aware that students can de-compile the class files, but the de-compiled code does not look anything like the original code and would be recognized by the professor if a student in a subsequent semester submitted the de-compiled code as their assignment solution.

TABLE II
PROGRAMMING ASSIGNMENTS

A1: ALU and Binary Tools Simulate an Arithmetic and Logic Unit; Write utility methods for working with binary numbers.
A2: Assembler Translate assembly language code into machine language.
A3: CPU Simulation Simulate a single-cycle, simplified ARMv8 datapath.
A4: Cache Simulation Simulate the cache memory and evaluate cache performance.

TABLE III
CONSTRUCTIVIST THEORY IN PRACTICE

Theory	Practice
Linking prior knowledge and experiences	Building upon knowledge from other courses; Programs implement concepts that were learned during class and practiced on lab assignments; The later parts of this project utilize components developed in earlier parts.
Scaffolding	Professor divides large project into parts; Professor provides guidelines, starter code, and tests for each part.
Seeking advice from an expert	Professor answers students' questions during class and office hours.
Collaboration with peers	Students are encouraged to work in pairs.
Authentic tasks	Creating a program that follows a specification; Understanding how to use pre-compiled code by reading the documentation; Choosing the appropriate programming approach.

The remainder of this section provides details about the four assignments that compose the semester-long project, as outlined in Table II. This project develops a Java simulation of a computer that implements a single-cycle, simplified version of the ARMv8 processor datapath as presented in the Patterson and Hennessy textbook [23]. This datapath supports nine assembly language instructions. One difference between the simulated computer in this project and the datapath in the textbook is that the textbook discusses 64-bit integers, while this project uses 32-bits for integers. Representing binary numbers using 32-bits simplified the use of Java for this project. Table III lists how the constructivist theory presented in Section II is applied in the programming project.

The assignment materials are available for download from the author's website: http://www.drcreilly.com/

1) A1: ALU and Binary Tools: In this assignment, students implement methods in the ALU (Arithmetic and Logic Unit) class and in the BinaryTools class. These classes are utilized in later assignments. The ALU class will be used in Assignment 3 as the ALU component of the processor datapath. The utility methods in the BinaryTools class are used in all of the

Decimal value 5 is 4-bit Binary value 0101
Represented in Java as an array of boolean type:

value	false	true	false	true
array index	3	2	1	0

Fig. 1. Java Representation of Binary Number as Array of boolean Type

future assignments. The learning goals of this assignment are to gain a deeper understanding of binary representation and arithmetic, and to review Java programming concepts. Students are provided with a test program that demonstrates the use of the ALU and BinaryTools classes, and runs a series of tests with output messages that indicate whether the correct output is produced by the methods that students are required to implement.

For the ALU class, there are seven private data members defined in the starter code: a data member for each of the two 32-bit binary input values, the control line input, the 32-bit output value, and the carry, overflow, and zero flags. Students are required to implement public methods that set the inputs and that get the output values. They also implement a public method that activates the ALU. The assignment information specifies that this activation method is to call the appropriate private method that performs the operation as specified by the control line value, and that the students must implement the operations on the binary numbers (they will lose points on the assignment if their code converts the values to decimal representation).

The BinaryTools class is implemented as a set of public static methods because the methods are conversion utilities and do not require the instantiation of an object. This class contains methods for converting between unsigned and signed decimal and binary representation, and methods that return a string representation of a binary number.

Discussions between the students and professor provided information about ways in which this assignment was useful for helping students expand their programming knowledge and review programming concepts. The programming concept that challenged many students was that the binary numbers in assignment are stored as a Java array of boolean type with the least significant bit in array index zero, as demonstrated in Figure 1. This means that the conceptual picture of the array is reverse from the typical Java concept of placing array index zero on the left. The idea that a Java array could be pictured with array index zero on the right appeared to be a big conceptual leap for some students. Another newer programming concept for many students was that they were required to follow the specifications without fully understanding how this Java class would be used in the overall computer simulation project. This provided an opportunity to experience the fact that in some cases a programmer must complete their assigned component without full knowledge of how that component will be used in the final product. For some students, Assignment 1 gave a much needed review of writing methods in Java, including: how to use method parameters, how to return the required value, and the difference between non-static and static methods.

TABLE IV
ASSEMBLY LANGUAGE INSTRUCTIONS SUPPORTED BY THE SIMULATED COMPUTER

Add values in Rm and Rn, put result in Rd: **ADD Rd,Rm,Rn**
Subtract values in Rn from Rm, put result in Rd: **SUB Rd,Rm,Rn**
Logical AND of values in Rm and Rn, put result in Rd: **AND Rd,Rm,Rn**
Logical OR of values in Rm and Rn, put result in Rd: **ORR Rd,Rm,Rn**
Load value from memory address (value in Rm plus literal offset) into Rd: **LDR Rd,[Rm,#N]**
Store value to memory address (value in Rm plus literal offset) from Rd: **STR Rd,[Rm,#N]**
Branch to label if contents of Rm are zero: **CBZ Rm,label**
Unconditional branch to label: **B label**
End of program (halt): **HLT**

2) A2: Assembler: For the second programming assignment, students complete methods in the Assembler java class. The assembler takes an assembly language program as input and outputs the corresponding machine language program. This Assembler class will be utilized in Assignments 3 and 4 to assemble the code that is run in the simulated computer. The assembler recognized the nine instructions that are supported by the simulated computer, as listed in Table IV.

Students are provided with starter code for the Assembler class, and are instructed to complete two methods in this class. Because the assembler is implemented as a two pass assembler, the first method that students must complete performs the tasks for the first pass and the second method that students complete performs the tasks for the second pass. Students are also provided with a Java class that tests the Assembler class using three input files with assembly language code that escalates in complexity.

This assignment reinforces concepts from this Computer Organization class, as well as concepts that span multiple courses in the major. At the same time as students are working on this assignment, the class has been covering the Assembly Language unit and completing small assembly language programs during lab. One of the challenges that students have with A64 assembly language programming are the requirements for how the A64 code must be formatted, which are much stricter than those students are accustomed to when writing Java code. Through the practice of parsing an assembly language code file in this assignment, students experience the reasons for the strict requirements for assembly language code format. The concept of a two-pass assembler will be familiar to students who have already taken the Programming Languages course as part of the major. Students who will take Programming Languages in the same or a future semester as this Computer Organization course will have the multiple-pass concept reinforced at that time. The Java programming concepts that are reviewed during this assignment include file I/O and string parsing. The professor reminded the class that there are multiple methods for performing these tasks in Java and

allowed students to choose their own approach for these tasks. Students are also reminded that the Java API documentation is an appropriate reference for obtaining information about how to accomplish these tasks in Java. For students who are at an earlier point in the major, this may be the first time that the professor has not provided specific instructions for how to perform tasks in Java.

3) A3: CPU Simulation: In Assignment 3, students complete methods in the CPU class in order to implement the fetch, decode, and execute steps of the machine cycle as a simulation of the single-cycle processor datapath. This is the most complex of the four assignments that compose the simulated computer project because it involves translating the datapath diagram that is discussed in class and presented in the textbook into Java code. Because of the complexity of this assignment, the professor dedicated class time to discussions of this assignment during the period while students were actively working on the assignment. These discussions were directed by the students' questions, and feedback received from students indicated that these in-class discussions provided a useful learning opportunity.

Students are provided with starter code for the CPU class, and with a test program that uses the Assembler class from Assignment 2 to assemble a program into machine language, then calls a CPU class method to run the machine language program. A JAR file is provided that contains Java class files for simulations of the ALU (from Assignment 1), the Assembler (from Assignment 2), and classes that represent other components within the CPU including the register file, memory units, multiplexors, and control units. Students are given documentation that describes the public methods of the classes that are contained in the JAR file, but they do not have access to the exact Java code for these classes.

This assignment directly addresses the concept of a single-cycle datapath including ideas about the timing of events within the cycle and the fact that this datapath trades some unnecessary work within the datapath in order to reduce the complexity of using a single datapath for executing different types of instructions. The concept of abstraction is also addressed by this assignment through the use of CPU components that have a known interface but an unknown implementation.

4) A4: Cache Simulation: The final assignment of the computer simulation project was to implement the cache memory unit that was used by the processor in Assignment 3. The single cache memory Java class uses parameters that are passed to the constructor to create an object that operates as a direct-mapped cache, set-associative cache, or fully-associative cache. Because of time limitations in the semester, the students only implement the memory read method. The test program that was provided to the students creates caches with different mappings and different configurations and runs performance experiments on the different caches. In addition to implementing the cache memory class, this assignment also required students to answer conceptual questions about the cache performance results.

TABLE V
STUDENTS ENROLLED IN STUDY COURSE

	Spring 2018	**Fall 2018**	**Overall**
Enrolled in Course	15	13	28
In First Two Years of Study	7 (47%)	2 (15%)	9 (32%)
Use Masculine Pronouns	12 (80%)	6 (46%)	18 (64%)
Participated in Survey	9 (60%)	6 (46%)	15(54%)

This assignment provides an opportunity for students to practice concepts related to cache memory, to experience computer systems performance testing, and to see how Java constructor parameters can be utilized to create a class that can be instantiated to perform in different ways.

V. SURVEY DESIGN AND RESULTS

This section presents the survey that was used to gather student feedback about the programming assignments, discusses the results of the survey, and identifies limitations with this study.

A. Methodology

This study was qualitative in nature and sought to further understand the phenomenon of collegiate emerging adult learning in a computer science education setting which employed constructivist methodology. The research utilized case study methodology and relied primarily on steps supported by the literature [24], [25]. This case study serves as a foundation for explorations of the broader phenomenon of effectively instructing computer science in a world of ever changing, fast-paced, resource-rich collegiate learning environments.

The participants in this study were emerging adult college students enrolled in the Spring 2018 and Fall 2018 sections of an Introduction to Computer Organization course at a liberal arts college in the northeastern United States. The range of ages for this sampling matches the generalized target of higher education students 18-25 years old [24]. Of the 28 students enrolled in these two offerings of the course, 9 students (32%) were enrolled in their first two years of study, and 18 of the students (64%) use masculine personal pronouns. Table V provides details about the students enrolled in the course.

Data was obtained by a survey consisting of a mixture of fifteen structured and open-ended questions, as shown in Tables VI and VII. The participants were informed that their survey responses were anonymous. The decision to administer an anonymous survey is supported by research that has shown that the associated reduction in social evaluative concerns and consistency expectations helps to minimize response bias [26]. Because the survey was designed to gather anonymous responses, the Institutional Review Board (IRB) that approved the survey required that the survey not include questions that could easily tie a response to a specific student. Therefore, questions about topics such as class year and gender identity were not part of this survey. The survey was administered anonymously using the web-based Qualatrics Software [27]. The survey participants provided direct self-report answers.

TABLE VI
STUDENT SURVEY AND SUMMARY OF RESPONSES (PART 1 OF 2)

1) Please indicate which courses you have taken prior to this course.				
	Spring 2018	*Fall 2018*	*Overall*	
Programming Languages (200-level)	5	6	11	
Design and Analysis of Algorithms (300-level)	3	3	6	
Computability, Complexity, and Heuristics (300-level)	0	0	0	
200-level CS Elective	1	1	2	
300-level CS Elective	2	1	3	
2) Please assess your programming ability throughout the semester. *(Choose: Poor = 1; Below Average = 2; Average = 3; Above Average = 4; Excellent = 5)*				
	Spring 2018	*Fall 2018*	*Overall Average*	*Overall Std. Dev.*
At the beginning of the semester, I thought my programming ability was	3.33	3.67	3.47	0.834
In retrospect, my actual programming ability at the start of the semester was	3.44	3.33	3.40	0.828
I would rate my current programming ability as	3.67	3.67	3.67	0.724
3) How useful was this class in developing your computer programming abilities? *Choose: Not at all useful = 1; Slightly useful = 2; Moderately useful = 3; Very useful = 4; Extremely useful = 5*				
	Spring 2018	*Fall 2018*	*Overall Average*	*Overall Std. Dev.*
	3.22	3.83	3.47	0.915
4) How frequently did you work with a partner during this course? *Choose one:*				
	Spring 2018	*Fall 2018*	*Overall*	
I worked with a partner during all of the programming assignments	33.3%	50%	40%	
I worked with a partner during some of the programming assignments	22.2%	16.7%	20%	
I did not work with a partner on any of the programming assignments	44.4%	33.3%	40%	
5) Rate the usefulness of working with a partner during this course: *Choose: Not at all useful = 1; Slightly useful = 2; Moderately useful = 3; Very useful = 4; Extremely useful = 5*				
	Spring 2018	*Fall 2018*	*Overall Average*	*Overall Std. Dev.*
	3.63	3.67	3.64	1.28

Face and content validity for the survey were established by having collegiate faculty confer and establish consensus on the questions in the survey. The results show consistency in responses across the two semesters. As detailed in Tables VI and VII, for the Likert-scale type of questions, the average response for each semester is within one standard deviation of the average response for the data from all semesters combined.

The students who were enrolled in these two sections of the course were invited to voluntarily participate in the survey after their semester grades had been released. During class near the end of the semester the professor described this study to the students and informed them that they would receive an email with the participation invitation after the end of the semester. The survey was open for two weeks, with a reminder email sent to the students at the beginning of the second week. Overall, fifteen students (54%) chose to participate in the survey, with a 60% response rate from the Spring 2018 class and a 46% response rate from the Fall 2018 class, as detailed in Table V.

B. Results

The two goals of the programming assignments in this Introduction to Computer Organization course were for students to gain experience with the concepts from the course, and for students to gain additional programming experience by completing a multi-part project during the semester. The survey results that relate to each of these goals are discussed in this section, followed by a discussion of the open response question.

The survey results were analyzed in two ways: first with all responses together, and second with responses divided into categories based on the other computer science courses a student has taken. Question 1 (indicate what other CS courses you have taken) was utilized to place the respondents into three groups based on their level of experience within the CS major. The three groups are: low experience (selected none of the courses, 4 respondents); medium experience (Selected only 200-level Programming Languages course, 5 respondents); and high experience (selected 200-level Programming Languages and 300-level Analysis of Algorithms courses, 6 respondents). Recall that the CS2 course is a prerequisite for this Computer Organization course, so even the students in the low experience group have previously taken one or two semesters of CS courses at this college.

1) Results Related to Programming Ability: In Question 2, students provide a self-rating of their programming ability at the beginning and the end of this semester. Question 3 asked students to rate the usefulness of this course in developing their programming abilities. Figure 2 shows the Likert-type scale responses to Questions 2 and 3 from the survey. Overall, students reported that their programming ability improved over the course of the semester. 40% (6 out of 15) of the students reported that they thought their programming ability at the beginning of the semester was above average or excellent. The second part of Question 2 asks students to reflect in retrospect on their programming ability at the beginning of the semester. With this reflection, 33% (5 out of 15) students reported that they believe their programming ability at the beginning of the semester was actually above average or excellent. Overall, students' self-rating of programming ability improved throughout the semester, with 53% (8 out of 15) of students reporting that their programming ability at the end

TABLE VII
STUDENT SURVEY AND SUMMARY OF RESPONSES (PART 2 OF 2)

6) Indicate your ability level for accomplishing the following tasks after completing this course: *(Choose: Very Low = 1; Below Average = 2; Average = 3; Above Average = 4; Very High = 5)*				
	Spring 2018	*Fall 2018*	*Overall Average*	*Overall Std. Dev.*
Write programs in Java	3.89	4.33	4.07	0.799
Convert numbers between number systems	4.56	4.50	4.53	0.743
Explain why binary computers use binary representations	4.33	4.50	4.40	0.632
Know the role of each component of the processor (ALU, registers, cache)	3.89	4.17	4.00	0.756
Compare and contrast strategies for cache management	3.44	3.83	3.60	0.737
Explain the steps a processor takes when executing a program	4.00	4.33	4.13	0.743
Explain one model for distributed computing	3.56	3.50	3.53	0.990
Explain one model for distributed data storage	3.33	3.17	3.27	1.160
7) Rate the usefulness of the following project assignments: *Choose: Not at all useful = 1; Slightly useful = 2; Moderately useful = 3; Very useful = 4; Extremely useful = 5*				
	Spring 2018	*Fall 2018*	*Overall Average*	*Overall Std. Dev.*
Assignment 1 - Binary number tools and ALU	3.56	4.50	3.93	1.030
Assignment 2 - Assembler	3.56	4.67	4.00	1.000
Assignment 3 - Simulation of ARMv8 CPU	3.89	4.50	4.13	0.915
Assignment 4 - Cache Memory	3.67	4.67	4.07	0.799
8) Respond to the following statements concerning the programming assignments: *Choose true or false; results are percent true responses*				
	Spring 2018	*Fall 2018*	*Overall Average*	
The programming assignments helped me improve my Java programming skills	100%	100%	100%	
The programming assignments were too difficult	25%	0%	14%	
The programming assignments directly related to the material covered during the lecture	75%	100%	86%	
The programming assignments helped me gain a deeper understanding of the concepts that were covered during lecture	100%	100%	100%	
9) The most helpful part of this class was: *Open response discussed in Section V-B3*				

of the semester was above average or excellent. Notably, at the end of the semester, all students reported that their current programming ability is average or higher. Question 3 asked if this was a useful course for developing programming ability. 53% (8 out of 15) students responded that this course was very useful or extremely useful, and no students rated the course as not at all useful. In Question 8, 100% of students responded with a true to a statement that the programming assignments helped them improve their programming skills.

When the responses are grouped based on the students' level of experience in the CS major, we find that the students in the high experience group provided a slightly higher self-rating of programming ability than those in the low experience group. There was at least one student in each of the experience groups who reported that their programming ability improved during the semester.

As reported in Question 4, 60% of the respondents worked with a partner for some or all of the assignments, while the other 40% never worked with a partner. Overall, students reported that it was moderately useful to very useful to work with a partner on the assignments.

2) Results Related to Course Content: In Question 6, students rated their ability level on various tasks that they learned during the semester. The average rating for the ability to accomplish the various tasks was Average to Above Average. Students gave a slightly higher rating of their ability to work with binary numbers than with other tasks. This seems appropriate because binary numbers were first discussed early in the semester, and the use of binary numbers throughout the semester provided many opportunities for the reinforcement

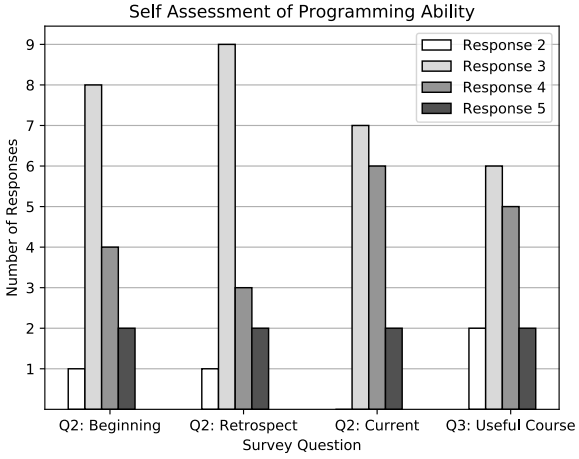

Fig. 2. Self Assessment of Programming Ability and Usefulness of Course in Developing Programming Ability

of this concept. The slightly lower ability ratings for cache management and distributed systems corresponds with these topics being covered near the end of the semester and not being covered as in depth as other topics.

Question 7 asks students to rate the usefulness of the four programming assignments. Students rated the all of the assignments as moderately useful to very useful. Assignment 3 was rated as slightly more useful than others. This seems appropriate because the topic of this assignment was a large focus of the middle of the semester. Assignment 1 and Assign-

ment 2 focused on more simple concepts, and were building blocks for Assignment 3. A large purpose of Assignment 1 is to show students what Java programming skills are needed for this class, and students who have adequate Java skills should find Assignment 1 somewhat boring. The professor notes that a couple of students withdrew from the class after Assignment 1 was assigned, and indicated that they would improve their Java programming skills before enrolling in this class during a future semester, demonstrating that Assignment 1 is serving its Java skills purpose.

On Question 8, all respondents indicated that the assignments helped them gain a deeper understanding of the concepts from lecture. The respondents in the high experience group all selected false on the statement that the assignments were too difficult, and all selected true on the statement that the assignments directly related to lecture. This is in contrast to the low experience group where half selected true and half selected false for each of these statements. These differences between the experience groups deserves further investigation after responses have been collected from more semesters.

3) Open Response Results: Question 9 asked the students to identify the most helpful part of this class, and provided space for an open response. A total of 10 students responded to this question, with 6 responses from the Spring 2018 semester and 4 responses from the Fall 2018 semester. The responses fall into three categories: the programming assignments, the course topics, and interactions with the professor.

Six students responded that the programming assignments were the most helpful part of the class. Four of those responses were a simple statement that identified the assignments. The two responses that provided more detail are:

- Labs and programming assignments that were based directly off the topics discussed in class.
- The later programming assignments (3 and 4). The first two did not serve a purpose for further understanding material, especially the first programming assignment. It just felt tedious as I filled out all the methods that needed to be completed. PAs 3 and 4 had a clearly defined purpose: 3 significantly helped me understand the process of the CPU, and 4 cleared up how memory works, although there could be room for improvement on PA 4.

One student identified the specific topics of assembly language and program execution as the most helpful part of the class.

Three students stated that interactions with the professor were the most helpful part of the class. These responses are:

- The professor was extremely helpful and helped me a lot in both studying for this class and overall java programming.
- Professor's office hours when students would often ask questions in a group and work through problems
- On paper, probably the programming assignments, but I think it's hard to overestimate the influence of a positive and enthusiastic teacher, who is open to questions (which I am happy to say I had)

In these responses, students identified constructivist practices that helped their learning. The students whose responses related to the programming assignments found that linking prior knowledge and experiences were most helpful. Those whose responses relate to interactions with the professor found that seeking advice from an expert was most helpful. Additionally, students state that these assignments built and reinforced their Java programming knowledge.

C. Limitations

One limitation of this study is the small number of survey responses. The small class sizes at this liberal arts college provide a small pool of students to participate in this study. However, the survey results can serve as a reference for which other computer science programs can apply generalized findings due to similarity in size and goals.

A second limitation of this study is that the data is collected from students' self-reported abilities and learning. Students self-assessments may or may not provide a true indication of learning. However, one of the goals of constructivist pedagogy is to build the learner's self-efficacy. Therefore, the self-reported learning data is an indication of self-efficacy. Furthermore, in addition to indicating self-efficacy, self-reported learning is also an indication of establishing proficient performance partly due to increased self-regulatory skills that are built through taxing activities. [26].

VI. Conclusions and Future Work

The professor's observations during the semester and the students' survey responses indicate that the programming project is meeting its goals for helping students learn the Computer Organization topics, and for providing an opportunity for students to gain experience with Java programming.

The survey results indicate that the gains in programming ability and knowledge about course content were directly related to the constructivist process in which these students built upon previous knowledge and experiences. The students started with a base-level of knowledge, due to completing the prerequisite CS2 course. As this Computer Organization course progressed, each student engaged in projects that required deep inquiry within a socio-cultural framework. The students had opportunities to connect and collaborate with peers on authentic tasks that were scaffolded in both content and process. Students reported gains in both programming ability and knowledge. These gains reflect both a deeper understanding (learning) and increased self-efficacy.

This study will continue during future semesters when this Computer Organization course is offered. The additional survey responses that are gathered during future semesters will provide an opportunity for deeper analysis of the impacts of this programming project on student learning. Additionally, this paper provides a detailed description of the programming projects and the study methodology so that others can replicate this study.

ACKNOWLEDGEMENTS

This research was partially funded by generous support from the Lubin Family Foundation.

REFERENCES

[1] Joint Task Force on Computing Curricula, Association for Computing Machinery (ACM) and IEEE Computer Society, *Computer Science Curricula 2013: Curriculum Guidelines for Undergraduate Degree Programs in Computer Science*. ACM, 2013.

[2] J. A. Bernauer and L. A. Tomei, *Integrating pedagogy and technology: Improving teaching and learning in higher education*. New York, New York, USA: Rowman & Littlefield, 2015.

[3] L. Vygotsky, *Mind in society: The development of higher psychological processes*. Harvard University Press, 1980.

[4] J. Piaget, *The construction of reality in the child*. Routledge, 2013.

[5] J. Brown, A. Collins, and P. Duguid, "Situated cognition and the culture of learning," *Educational Researcher*, vol. 18, no. 1, pp. 32–42, 1989.

[6] A. Bandura, *Social foundations of thought and action: A social cognitive theory.*, ser. Prentice-Hall series in social learning theory. Englewood Cliffs, NJ, USA: Prentice-Hall, Inc., 1986.

[7] J. Bruner, *On knowing: Essays for the left hand*. Cambridge, MA, USA: Harvard University Press, 1962.

[8] G. Wiggins and J. McTighe, *Understanding By Design*. Association for Supervision & Curriculum Development, 2005, ch. Backward Design, pp. 13–34.

[9] J. Bruner, *The Culture of Education*. Harvard University Press, 1996.

[10] S. Papert, *Mindstorms: children, computers, and powerful ideas*. New York, NY: Basic Books, Inc., 1980.

[11] L. Barker, C. L. Hovey, and L. D. Thompson, "Results of a large-scale, multi-institutional study of undergraduate retention in computing," in *2014 IEEE Frontiers in Education Conference (FIE) Proceedings*. IEEE, Oct 2014, pp. 1–8.

[12] O. Hazzan, T. Lapidot, and N. Ragonis, *Guide to Teaching Computer Science: An Activity Based Approach*, 2nd ed. London: Springer-Verlag, 2014, ch. 2, pp. 15–22.

[13] S. Freeman, S. L. Eddy, M. McDonough, M. K. Smith, N. Okoroafor, H. Jordt, and M. P. Wenderoth, "Active learning increases student performance in science, engineering, and mathematics," *Proceedings of the National Academy of Sciences*, vol. 11, no. 23, pp. 8410–8415, 2014.

[14] N. Jumaat, Z. Tasir, N. Abd halim, and Z. Mohamad Ashari, "Project-based learning from constructivism point of view," *Advanced Science Letters*, vol. 23, no. 8, pp. 7904–7906, October 2017.

[15] R. A. Knuth and D. J. Cunningham, "Tools for constructivism," in *Constructivism and the technology of instruction: a conversation*, T. M. Duffy and D. H. Honasses, Eds. Hillsdale, NJ: Lawrence Erlbaum Associates, 1991, pp. 163–187.

[16] I. Jung and Y. Suzuki, "Scaffolding strategies for wiki-based collaboration," *British Journal of Educational Technologies*, vol. 46, pp. 829–838, 2015.

[17] D. Taipala, "Teaching computer architecture in an online learning environment using simulation and peer instruction," *J. Comput. Sci. Coll.*, vol. 30, no. 1, pp. 87–98, Oct. 2014. [Online]. Available: http://dl.acm.org/citation.cfm?id=2667369.2667385

[18] G. S. Wolffe, W. Yurcik, H. Osborne, and M. A. Holliday, "Teaching computer organization/architecture with limited resources using simulators," *SIGCSE Bull.*, vol. 34, no. 1, pp. 176–180, Feb. 2002. [Online]. Available: http://doi.acm.org/10.1145/563517.563408

[19] C. Norris, J. B. Fenwick, Jr, J. Wilkes, and K. H. Jacker, "Blending object-oriented design principles and software engineering practices into an undergraduate architecture simulator project," in *Proceedings of the 43rd Annual Southeast Regional Conference - Volume 1*, ser. ACM-SE 43. New York, NY, USA: ACM, 2005, pp. 329–334. [Online]. Available: http://doi.acm.org/10.1145/1167350.1167443

[20] L. Ivanov, "A hardware lab for the computer organization course at small colleges," *Journal of Computing Sciences in Colleges*, vol. 19, no. 2, pp. 185–190, December 2003.

[21] N. Sprague, "Arduino as a platform for a computer organization course," *Journal of Computing Sciences in Colleges*, vol. 28, no. 3, pp. 53–60, January 2013.

[22] M. B. Gousie and J. D. Teresco, "Helping students understand the datapath with simulators and crazy models," in *Proceeding of the 44th ACM Technical Symposium on Computer Science Education*, ser. SIGCSE '13. ACM, 2013, pp. 329–334.

[23] D. A. Patterson and J. L. Hennessy, *Computer Organization and Design ARM Edition: The Hardware Software Interface*. Morgan Kaufmann, 2016.

[24] J. W. Creswell, *Educational Research: Planning, Conducting, and Evaluating Quantitative and Qualitative Research*, 4th ed. Upper Saddle River, New Jersey: Pearson, 2015.

[25] R. E. Stake, *The Art of Case Study Research*. Sage Publications, 1995.

[26] A. Bandura, *Self-Efficacy Beliefs of Adolescents*. Information Age Publishing, 2006.

[27] "Qualtrics," https://www.qualtrics.com, 2018.