Computer Science Principles: Impacting Student Motivation & Learning Within and Beyond the Classroom

Kara A. Behnke
University of Colorado Boulder
ATLAS Institute, 1125 18th Street
320 UCB, Boulder, CO 80309
Kara.Consigli@Colorado.edu

Brittany Ann Kos
University of Colorado Boulder
ATLAS Institute, 1125 18th Street
320 UCB, Boulder, CO 80309
Brittany.Kos@Colorado.edu

John K. Bennett
University of Colorado Denver
inWorks, 1380 Lawrence St.
Suite 1400, Denver, CO 80204
JKB@UCDenver.edu

ABSTRACT

The Computer Science (CS) Principles framework seeks to broaden student participation and diversity in the field by focusing on the creative and social aspects of computing. As the pilot effort undergoes its early execution phases, this research contributes to the theoretical and practical application of CS Principles. We investigated the impact of CS Principles on student motivation and learning outcomes and sought to determine if the pedagogy created any lasting change on student perceptions of CS as a field of practice.

We report a case study of how CS Principles created an effective framework for introducing undergraduate students to the fundamentals of computer science. We discuss how Self-Determination Theory instantiates Self-Directed Learning, Constructionist, and Connectivist learning theories, which can be used to inform the pedagogical framework. Quantitative and qualitative measures were used to assess the impact of CS Principles on student motivation and learning outcomes, followed by an additional surveying of students one year after the completion of the course.

Results indicate that CS Principles facilitated positive programming experiences for students, helped increase learning interest and improve attitudes of CS as a field of study, positively changed perceptions of CS as a creative practice, and also encouraged students to continue learning CS after the course had finished. In particular, many non-majority students in the course self-reported to having positive changes and attitudes about CS explicitly because of the course. These finding suggest that CS Principles is a step in the right direction for creating more engaging and compelling curricula to diverse groups of students, especially those with minimal experience and exposure in the field. We discuss opportunities for future work using the selected theoretical framework for CS Principles.

Keywords

Computer Science Principles; computational thinking; self-determination theory; self-directed learning; constructionism; connectivism; motivation; learning outcomes.

1. INTRODUCTION

As educators, we want to create meaningful and engaging learning experiences for computer science (CS) students. As researchers, we want to find empirical evidence for what *exactly* fosters meaningful and engaging experiences in the learning process. How should we teach effective and foundational CS skills that students need to succeed in the 21st century, yet provide a meaningful and engaging curricula that appeals to them?

Although the computer science community has labored to increase participation in K-12 education [28, 7] and to create intellectually rich and engaging courses [29, 19, 38, 17, 23, 24, 18, 28, 6], many students, particularly females and students of color, find traditional approaches for learning computer science off-putting, asocial, and boring [25, 4, 16]. Some consider Advanced Placement[1] examinations as a bridge to help students engage and enroll into CS courses in higher education [16]. Although the number of students taking Advanced Placement (AP) CS courses has increased in recent years, the number of students taking this examination is much smaller compared to other science, technology, engineering and mathematics (STEM) AP Examinations [16]. These efforts show that teaching introductory CS on a broad level and in an engaging way, particularly for those with limited exposure or no previous experience in CS, remains a significant pedagogical challenge.

Computer Science Principles is a new AP computer science course designed to introduce students to the central ideas of CS, to instill ideas and practices of computational thinking [40], promote programming literacy and creativity, and to engage students in activities that show how CS impacts and changes the world [5, 3]. CS Principles is intended to help students gain competencies similar to those gained by students completing a university CS course for non-majors [3].

The Computer Science Teachers Association (CSTA) Standards form a basis by which educators can begin to implement a coherent CS curriculum that is available to all students [5], but more work needs to be done in designing, implementing, and evaluating these standards in real-world classrooms [3]. For example, although pilot projects have tested how CS Principles could support different teaching learning approaches and portfolio-based assessments for the AP examination [3], research remains limited on whether students who have taken CS Principles continue to pursue computing by taking additional CS courses, or if they pursue computing in other venues after course completion. In addition, we have limited understanding how different populations undertake and perform in CS Principles, despite its primary goal to broaden participation and diversity in computing [35, 3].

[1] The Advanced Placement (AP) program is a rigorous academic program in the United States that allows high school students to take college-level courses and exams and to potentially earn college credit while still in high school.

ICER '16, September 08-12, 2016, Melbourne, VIC, Australia.

DOI: http://dx.doi.org/10.1145/2960310.2960336

Previous work suggests the CS Principles framework can potentially influence student engagement for learning introductory computer science [5, 3]. This research extends prior work by examining the impact of CS Principles on student motivation and learning outcomes while in the classroom and after one-year of course completion. The research questions to be addressed through the study were: (1) *Does Computer Science Principles improve student motivation in learning CS?* (2) *Does Computer Science Principles meet the intended learning objectives of the framework?* (3) *Do these results differ across differing student populations?*

We begin introducing the relevant literature related to CS Principles and learning engagement, detail the theoretical framework and case-study of measuring student motivation and learning outcomes in a CS Principles undergraduate course in the United States, and conclude with a discussion of future opportunities in CS Principles pedagogy.

2. BACKGROUND

In this section we review the current literature on CS Principles pedagogy and its connection to human motivation research and educational research. We explore the application of Self-Directed Learning, Constructionism, and Connectivism into both the design and facilitation of the CS Principles framework.

2.1 Computer Science Principles

Computational thinking [41] captures many important aspects of the work in which computer scientists engage, including using algorithmic approaches to solve challenging problems. Therefore, CS Principles is organized into seven *Big Ideas*[2] with six correlating *Computational Thinking Practices*[3] [5]. The *Big Ideas* represent the fundamental CS skills that are essential to succeed in future college courses and in a variety of computing and STEM careers [5]. The *Computational Thinking Practices* represent foundational computer science theories and practices [5]. The CS Principles framework constructs thirty-five curriculum learning objectives by matching the seven big ideas with the six computational thinking practices [9].

Pilot implementations of CS Principles courses [35, 3], taught at both high school and collegiate levels appear to have positively impacted females and underrepresented students [3], but these pilots differed in structure, breadth, and focus [35]. One possible reason for this is that the CS Principles content, represented by the framework's list of topics and objectives and its broad assertions of desired outcomes, resulted in multiple interpretations by instructors to achieve the learning outcomes [35].

The vision for CS Principles is not a packaged curriculum ready to be taught and administered, but rather a curriculum framework specified by the seven loosely defined big ideas and the six computational thinking practices. Therefore, pilot CS Principles efforts reflect a broad interpretation of the big ideas and computational thinking practices, resulting in substantially different courses [35]. Even though most of the pilot sites reported that students found the curriculum engaging [35], more evaluation is needed to determine whether students achieved the desired learning outcomes intended for CS Principles.

Pilot efforts have also shown how CS Principles can support different teaching and learning approaches and portfolio-based assessments for the AP examination [3], yet there is limited research on whether students who have taken CS Principles continue to pursue computing by taking additional CS courses, or if they pursue computing in other venues after course completion. Broadening participation and diversity in computer science is a key objective and helped initiate the creation of the CS Principles framework [5, 9], yet more work is needed to determine if there is a meaningful impact on students, particularly for non-majority populations, beyond the course itself. Thus, prior work leaves considerable room to explore other approaches to teaching introductory CS to diverse groups of students and to assess how these approaches may potentially impact their motivation and continued pursuit of CS education.

The primary goal of this research is to examine whether CS Principles impacts student motivation and intended learning outcomes for teaching the fundamentals of computer science. The secondary goal of this work is to explore any potential lasting effects on student learning after course completion. In order to conduct this work, a theoretical lens is needed that brings into focus the disparate ideas of effective CS pedagogy and human motivation research. The conceptual framework needs to (1) frame the pedagogical outcomes that are desirable by the CS education community, and (2) use a theory that is appropriate for the design and implementation of effective motivational learning techniques. As explained below, the conceptual and theoretical framework chosen for this work is Self-Determination Theory [32].

2.2 Self-Determination Theory

Self-Determination Theory (SDT) is an empirically derived theory of human motivation that distinguishes different types of motivation based on the reasons or goals that give rise to an action. The most basic distinction of human motivation is between *intrinsic motivation*, which refers to doing something because it is inherently enjoyable or interesting, and *extrinsic motivation*, which refers to doing something because it leads to a desired outcome [31]. Much theoretical discussion has postulated whether extrinsic or intrinsic motivation is better for learning [31]. SDT postulates there are three universal psychological needs that are essential for optimal human development and functioning—competency, autonomy, and relatedness [31]—and that these can arise due to both intrinsic and extrinsic factors [33].

SDT has shown that people learn more effectively and creatively when they feel competent, autonomous, and related to something bigger than themselves [31]. We present a novel instantiation of autonomy, competency, and relatedness that build upon other learning theories: Self-Directed Learning [20], Constructionism [29], and Connectivism [34]. In the spirit of computational thinking, we present Self-Directed Learning, Constructionism, and Connectivism as a subclass of SDT that inherit *autonomy*, *competency*, and *relatedness*. The relationship of this connection to CS Principles is discussed further in the remainder of this section.

2.2.1 Self-Directed Learning—Autonomy

Autonomy is our innate desire to be self-directed. Autonomy within SDT concerns a sense of volition or willingness when doing a task [31, 21, 32]. When activities are done for interest or

[2] Big Ideas: Creativity, Algorithms, Programming, Abstraction, Data and Information, The Internet, and Global Impact

[3] Computational Thinking Practices: Connecting Computing, Analyzing Problems & Artifacts, Developing Computational Artifacts, Abstracting, Communicating Collaborating

personal value, perceived autonomy is high. Ryan et al. [33] argue that, "Provisions for choice, use of rewards as informational feedback (rather than to control behavior), and non-controlling instructions have all been shown to enhance autonomy and, in turn, intrinsic motivation" [33]. SDT thus shows how learning can enhance autonomy when learning provides students with flexibility, choices in how to select tasks and complete goals, and use reward structures to provide meaningful feedback [33]. Self-Directed Learning [20] may also inform ways to design motivational structures into pedagogy.

In this research, we incorporate self-directed learning into the CS Principles framework. We attempt to implement the six principles of self-directed learning into the curriculum and content design. In the case study, students are encouraged to (1) *take initiative* regarding the specific content on which to focus their efforts so that they can (2) take responsibility for their learning by *selecting and assessing* their own learning activities, which can potentially (3) sustain their *motivation* and *volition* because of the emphasis on individual creative interests, and (4) by *setting their own goals*, which are framed carefully in the learning environment in order to (5) help *scaffold* these objectives and skills for students, so that they can (6) create peer-to-peer *collaboration* within their learning communities. In this work, course curriculum is built around these principles in an effort to teach the process of computational thinking by encouraging and facilitating students with "self-directed autonomy" throughout the CS Principles framework.

2.2.2 Constructionism—Mastery

In SDT, competency refers to our innate desire to gain *mastery*[4] of an activity over time, i.e., to get better and better at what we do [31, 10, 32]. SDT further contends that mastery, or the process of gaining competency in self-ability, is essential for education and motivation [31, 32]. Mastery of an activity is challenging, hard work [14, 15], but gratifying through induced flow [10]. Mastery creates the sense of accomplishment and efficacy, which is facilitated by the exercise of one's capacities under conditions of optimal challenge [32]. These elements promote flow [10] by allowing learners to overcome failures and master the material. This work also leads CS educators to employ "learning by doing" pedagogy, particularly through the lens of constructionism.

Constructionism [29], a subclass of constructivist theory [28], posits that learners construct mental models to understand the world around them. In contrast to constructivism, constructionism argues that learning happens most effectively when students are also active in making tangible artifacts in the real world to reflect their understanding of the learned material. Therefore, constructionism has often been used to teach computer science [19, 29]. By actively constructing tangible artifacts to demonstrate understanding, students learn the material through mastery and persistence. Constructionism facilitates computational thinking [8] and computational artifacts through the recursive learning [25] of mastering a challenge.

This research posits that CS Principles incorporate constructionism when students actively construct models, computational artifacts, and computer programs. We use constructionism in this model to facilitate "learning-by-making" [19] in an effort to make CS more personally engaging, motivating, and meaningful for introductory students [23, 24, 3]. This appears to be an effective approach, since Brown [8] and Wagner [39] argue that innovation-creation and computational thinking can only be effectively learned within iterative constructionism. Having the desire to master something because one finds it deeply satisfying and personally challenging inspires the highest levels of creativity [2], which is a direct goal of the CS Principles [3].

2.2.3 Connectivism—Purpose

Feelings of relatedness to community, or *purpose*[5], is essential for optimal human development and well-being [31]. Relatedness and purpose concern the emotional and personal bonds between individuals. It reflects our strivings for contact, support, and community with others [32]. Purposeful needs are not antithetical to autonomy or mastery; in fact, one often feels most related and purposefully driven to those who are responsive to one's autonomous expressions [31, 32]. When students experience support of their autonomy and feel connected to and supported by others, they are more likely to be highly motivated in the learning process [33]. Students are engaged and motivated in domains where they feel purposefully connected [31, 33], which gives rise to learning theories that enable learning experiences to be conducive toward interrelation and purpose.

Connectivism [34] guides the development of learning materials for the networked world. Connectivist theory asserts that individuals learn and work in a networked environment [34]. As a result, learners do not have control over what they learn since others in the network continually and actively change information, requiring new learning, the unlearning of old information, and relearning current information [1]. However, this "loss of control" does not indicate a loss of autonomy for the learner—rather, connectivists advocate for enabling learners to become autonomous and independent so that they can retroactively acquire information to build a valid, accurate, and current knowledge base [1]. Some of the key intellectual ideas behind digital learning includes connectivism (learning is the connection of people to knowledge), distributed representation (knowledge exists in the network of learners) and negotiated meaning (the meaning of knowledge is negotiated in the network) [12, 13]. Connectivism is fundamentally driven by individuals creating purpose and communal meaning within a network. Therefore, connectivist communities of purpose increasingly play an important role for human motivation and learning.

In summary, under conditions conducive to autonomy, mastery, and purpose, people will likely express their inherent tendency to learn, to do, and to grow. People are engaged and motivated in domains where their basic psychological needs can be fulfilled [32]. These interpersonal processes are fundamental to learning and personal motivation. Self-directed autonomy, constructing mastery, and connectivist purpose thus inform the design and application of this instantiation of Computer Science Principles.

3. METHOD OF STUDY

3.1 Purpose

To answer the three research questions, we conducted a case study [36] within an introductory computer science undergraduate

[4] We use "mastery" nomenclature over Ryan & Deci's [31] "competency"; mastery underscores an ongoing process of learning, whereas competency suggests a completed state of experience or being [10].

[5] We use "purpose" as a term for Ryan and Deci's [31] "relatedness"—*purpose* can refer to multifaceted forms of relatedness, rather than just interpersonal connection [32, 11].

course. We used a mixed methods approach that examined both quantitative and qualitative data. We then examined whether results differed across student demographics, gender, etc., and if so, the way in which the results differed. We investigated how CS Principles pedagogy can support self-directed learning, constructionism, and connectivism, while also exploring whether there was a lasting impact of the course on students one-year after the completion of the material.

3.2 Research Design

3.2.1 Site Selection

An introductory CS course intended for undergraduate non-majors was chosen as the site of this case study. This course was historically taught in an interdisciplinary department at a public four-year university in the United States. The course's reputation from previous years—many different majoring students taking the course, and a higher representation of women compared to other introductory CS courses—provided a favorable demographical representation, which matches the theoretical objectives for the CS Principles pilot curricula and the intended research objectives. This introductory CS course was not a required course for any degree or major, although it did offer elective credit for departments in liberal arts and CS programs at the university.

3.2.2 Duration of Study

The duration of the case study was one academic semester (sixteen weeks). One year after the class took place, a follow-up survey was issued to students to measure any lasting effects or impact of the course.

3.2.3 Description of Population

A total of ninety-four students enrolled into the course. Twenty-seven people identified as female (29%). Although the representation of women in the course is not ideal, it is more than double the U.S. national average (12%) of undergraduate CS degree recipients who are women [4].

The ethnic distribution of students enrolled in the course were predominately White (77%). However, non-majority populations were also present, including American Indian or Alaska Native (4%), East Asian or Pacific Islander (11%), Middle Eastern or Central Asian (2%), Latino or Hispanic (4%), and Black or African American (2%). This diversity spectrum of non-majority students is generally similar to those who take AP CS courses in high school [16].

Thirty-four different majors were represented in the course, including majors in dance, film, psychology, mathematics, physics, engineering, biochemistry, neuroscience, political science, architecture, environmental design, among others. Twenty-five students (27%) were undeclared, open-option, or pre-engineering majors. Eleven students (12%) were declared CS majors. Most of the students taking this course were freshman (41%); however, a range of other students also took the course including sophomores to fifth-year seniors.

3.2.4 Implementation

3.2.4.1 Course Website

To facilitate the CS Principles framework, the course had a website with a carefully designed learning-management system (LMS) for the students to use throughout the semester. The course website served as the main platform for students to retrieve lesson materials and to upload their coursework. The course website was built using WordPress, an open-source content management system that uses HTML, CSS, PHP, and MySQL to manage Web content. The LMS also implemented user-experience mechanics to provide "self-directed learning," "constructionist," and "connectivist" mechanisms through the website. This topic is discussed further below.

3.2.4.2 Course Structure

The undergraduate course used CS Principles to structure the course [9]. A review of the computational thinking practices and their associating big ideas leads to an association of *Connecting Computing* and *Analyzing Problems and Artifacts* with self-directed learning mechanisms, *Developing Computational Artifacts* and *Abstracting* with constructionist-learning methods, and that *Communicating* and *Collaboration* practices with connectivist connections.

Connecting Computing and *Analyzing Problems & Artifacts* encourage self-directed learning, or student autonomy, by encouraging students to "learn to draw connections," to be "creative" in their learning, to "evaluate and analyze their own computational work," and "develop and express their own solutions" to computational problems [5, 9]. The focus on self-development and the freedom of creative expression underscores autonomy. *Developing Computational Artifacts* and *Abstracting* support constructionist learning or help students gain mastery of skills through "learning-by-making." For these computational thinking practices, students learn by "designing and developing interesting computational artifacts," and "develop models and simulations of natural and artificial phenomena," which leads to the "[facilitation of] the creation of knowledge" [5]. Students engage in constructionist learning by modeling and building artifacts, which encourages the student into the construction of knowledge. Finally, the *Communicating* and *Collaborating* practices support connectivist learning by requiring students to "describe computation and the impact of technology and computation" [5]. Students contemplate and communicate purpose, "collaborate on a number of activities" with other students, and promote a sense of relatedness by expressing how "computing has global impact" [5]. The connection and communication with others to create knowledge employs the foundation of connectivism. We used these computational thinking practices, theoretically framed by SDT, to structure the course objectives and intended learning outcomes for the course. These objectives were reflected in the course syllabus, course website, and learning objectives via assigned homework.

3.2.4.3 Course Learning Objectives

The course learning objectives directly correlate to the CS Principles computational thinking practices and big ideas. The computational thinking practices frame the "policy" or larger theoretical implications of the course, whereas the big ideas implement the "mechanisms" to facilitate these practices. Therefore, the curriculum required students to complete six CS Principles by finishing thirty-five laboratory assignments, six online quizzes, a midterm examination, self-reflections on their learning each week, and a final semester project. Each assignment for the course was grounded in self-directed learning, constructionism, or connectivism, and designed to reflect the core affordances of the relevant big idea and computational thinking practice. Students had to complete all laboratory assignments and online quizzes for each computational thinking practice in order to complete that learning objective for the course.

In addition to completing all six computational thinking practices, students had to take a midterm examination, post self-reflections about their learning progress, and complete a semester-long project. Rather than take a final examination, students were required to plan, develop, and submit a semester-long project by the end of the course. The semester project was intended to be broad enough to allow students to express individual creativity, and structured enough so that students produced approximately the same amount of work. The semester project required students to submit a Project Proposal, Digital Artifact (construction of a computational artifact), and Written Report (communicating purpose) of their work. Students also had opportunities to receive extra credit toward their grade by completing extra Labs.

3.3 Theoretical Framework

3.3.1 Self-Directed Autonomy

Self-directed learning was implemented into the curriculum by employing a "self-paced" structure for the course. Each of the six requirements enumerated by [20] was realized in a particular aspect of the course.

Students could choose where to put their efforts, within a hard deadline for all coursework at the end of the semester. In order to facilitate curriculum objectives in a self-paced course, all the course materials and assignments were available on the LMS website during the first week of class. We also implemented *scaffolding* techniques [20] to support student autonomy. This included setting semester-long goals for the students, but allowing enough flexibility for them to focus their efforts onto their own interests. The official learning objectives for the course were: (1) Demonstrate your mastery of six CS Principles, with each Principle requiring you to take an online Quiz for completion; (2) Research and develop a Semester Project; (3) Complete the Midterm Examination; (4) Write self-reflections about your learning experiences throughout the semester on the website Forums or submit privately on the course website.

For example, the semester project was open-ended so that students could focus on a particular topic that was of interest to them. In addition, the requirement for students to self-reflect about their learning experiences by writing about them was a mechanism designed to assess students' individual learning activities, a critical component to self-directed learning. Finally, students were given an opportunity to receive extra for completing additional assignments. Students could independently choose whether or not to complete these additional assignments, just as they could set their own goals by choosing which assignments to focus on first, last, etc.

The course website offered several ways to scaffold and guide student work, while preserving student autonomy.

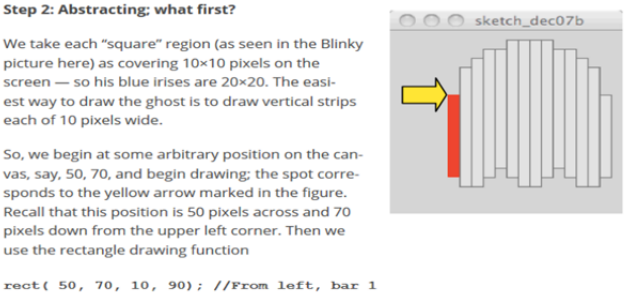

Figure 1. Step-by-step Scaffolding in a Programming Lab.

One of these scaffolding mechanisms was a class schedule with recommended deadlines and objectives on a week-to-week basis. Labs also provided appropriate scaffolding mechanisms to guide the student through the content. This was done by explicitly listing the objectives for that particular assignment, offering step-by-step guidance when appropriate, and by providing hints for the more complex or challenging lessons.

Figure 1 exemplifies how the LMS scaffolds students to complete the "Blinky *Programming* Lab," in which students draw the *Pac-Man*© character, Blinky, using the Processing programming language. Even though the "helpful hints" were provided at the end of the Lab, students could not simply look up the answer or write the code from what they saw on the lesson page. They had to analyze the information given to them, use that information to construct knowledge, and "think computationally" to solve the problem at hand. This scaffolding structure was implemented into every available Lab to help guide the students into taking responsibility for their own learning.

3.3.2 Constructing Mastery

Receiving frequent and relevant feedback on performance is an important process towards mastery. Therefore, we focused on providing students with feedback on the *progress* they made in completing the coursework. When students completed an assignment, they were not given a conventional grade (i.e., a student receiving an "A" or a 3.5 on a 4.0 scale or a 90% mark) but rather were given visual feedback on their progress through the website's user interface (see Figure 2). The website user interface incorporated a "progress bar" that highlighted the student's standing in the completion of course objectives. Figure 2 is a screenshot of the user interface depicting one way that students receive feedback on course progress.

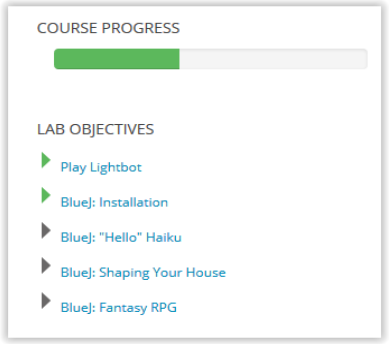

Figure 2. Mastery Feedback for Student Progress.

We chose to provide feedback on course progress so that students can adjust their behavior as needed and know what they need to do next to complete course objectives.

In an effort to prevent cheating, student feedback via the "progress bar" was updated only after the instructor or teaching assistant reviewed and "approved" the lesson submission. This practice allowed the instructor or teaching assistant the ability to "grade" student work, ensuring that students aren't cheating by submitting irrelevant or poor quality work. If student work was not up to standards, they could resubmit until they had mastered the material. This feedback employs the constructionist learning principle of "learning through effort." In addition, many Labs required learners to construct digital artifacts through code or tooling technologies, thus enabling the "constructionist mastery."

3.3.3 Connectivist Purpose

In an effort to create a sense of purpose and social engagement for students, the course website implemented a connectivist framework similar to conventional social media. Students had the ability to create avatars (upload a picture to their profile), update status on an "activity stream," the ability to "friend" other students or make peer-to-peer connections, communicate through private messaging and public chat forums, and even form "groups" to collaborate with specific sets of users.

Bulletin-board systems (BBS), more commonly known as forums, were also implemented into the course website to promote connectivism. BBS functionalities were implemented into the LMS to provide a mechanism for students to collaborate and share. Forums allowed students to post to specific "threads" to organize content. Students could also create private or public "groups" that invited specific people to that microcosm. We chose to implement these social media-like features in an effort to support and facilitate connectivist networks, which may potentially create a sense of purpose for students in the learning environment.

3.4 Instrumentation & Evaluation

3.4.1 Procedures

This case study implemented proxy pre-test and post-test design [30] to test the research questions. Primary data was collected in the form of quantitative and qualitative surveying, student course grades, website metadata, and content analysis. Proxy pre-tests and post-test responses measuring overall student interest in CS were analyzed, and triangulated by qualitative data survey responses.

3.4.1.1 Student Course Grades

Discrete evaluation in the form of course grades[6] were used as one source of quantitative data for analyzing learning outcomes. Student grades resulted in a non-normal distribution, which was skewed in a positive direction. Therefore, non-parametric tests were used to evaluate and analyze the grades as quantitative data. We triangulated student course grades to other qualitative data, since course grades themselves are often too coarse to give meaningful data regarding students' learning.

3.4.1.2 Proxy Pre-Test & Post-Test

Using the Intrinsic Motivation Inventory (IMI) is an established methodological practice for measuring motivation in Self-Determination Theory research [11]. The pre-test and post-test survey used the IMI questions and employed a five-point Likert scale to compute the dependent variable of "student motivation" into an ordinal, interval, and nominal data scales.

The survey instrument also included additional questions beyond the IMI scale in an effort to measure students' familiarity with technology, whether they had studied programming or computer science prior to the course, if they considered CS a creative practice, how they rated their pre- and post-programming skills, etc. These questions were theoretically aligned with IMI and computational thinking questions in order to determine overall interest, attitudes, and pursuits of computer science. Open and axial coding procedures were used to analyze open-ended questions. Summative and inferential statistics were used to analyze the quantitative questionnaire data.

3.4.1.3 One-Year Follow-up Survey

We conducted a one-year follow-up survey for students who participated in the undergraduate course. This survey was used to assess whether the findings in the pre-test and post-test were reflected in the students' actions or perceptions after the course had completed. For example, itemized questions asked if students continued to learn CS in traditional undergraduate courses, through free online tutorials, in non-traditional forms such as hack-a-thons, etc. The survey was emailed directly to students who had participated in the course the previous year. A total of twenty-six participants responded to the one-year follow-up study (28% response-rate of students whom participated in the course). The evaluation procedures for the follow-up survey were similar to the pre-test and post-test survey, including open and axial coding, and the use of summative and inferential statistics.

3.4.1.4 Content Analysis

Web analytics were collected as a primary metric of quantitative and qualitative observation. Information provided by participants through the website, such as user names, profile avatars, number of visits, number of comments, lesson submissions and time-stamps, among others, were represented through descriptive summaries. In addition, website information was included as part of the qualitative content analysis procedures, which included open coding and axial coding techniques.

3.4.1.5 Threats to Internal Validity

The principal researcher of this work was also the lead instructor for the course. Although the role of a *teacher researcher* [22] is a common methodological process in education research, we recognize that is less common in other fields of research, such as computer science education. Previous work [22] indicates that teacher research offers an opportunity to shape educational practice and to validate, affirm, and improve teachers' personal practice. In addition, the role of a teacher-researcher is arguably less problematic in the context of a case study, since case studies are, by definition, situationally subjective and bounded by time and activity, which is directly dependent on the context of its environment [36]. In this context, the role as teacher-researcher aligns with the case study research strategy.

Because the research site was an undergraduate course, students were not required to participate in the research. As part of the informed consent procedures, we emphasized that students' participation or lack of participation in the research would not affect their course grade. In addition, we did not analyze any research data until after the course had concluded for the semester. We incentivized students to participate in the research by providing them with extra credit towards their final grade for completing any research procedures. Students also had other opportunities to obtain extra credit that were not related to any research procedures (such as completing additional Labs and coursework).

Another potential threat to validity is that the one-year follow-up survey had a low response rate of 27%. Therefore, we recognize that these respondents might be biased, in that students who kept some connection with computing were also more motivated to

[6] Although students were not traditionally graded while enrolled in the course, the instructor needed to "convert" students' progress into course grades into the university system at the end of the semester. Final grades were calculated according to how many learning objectives they completed in the course total.

complete the survey. We discuss this in more detail in the Future Work section.

4. ANALYSIS & FINDINGS

Findings from this case study show that CS Principles is an effective framework for introducing students to the fundamentals of computer science. Results from the post-test and the one-year follow-up survey show that many students went on to pursue CS in some capacity. In addition, CS Principles positively influenced student perception of CS and attitudes about the field as a whole. Both quantitative and qualitative data showed that students perceive CS as a creative, socially relevant, and important field of practice.

4.1 Impact on Learning Outcomes

Course grades were used as a measure for the dependent variable "learning outcomes." The majority of students did well in the course, where forty-two students (45%) received A's[7] and 18% of students received B's. Since the CS Principles curriculum seeks to appeal to a broad audience, we used a Spearman's Rho test to measure if there was any correlation between student gender and student grades in the course. The association between grade and gender was found to not be statistically significant; therefore, students in the variable group did not fare better or worse in the course strictly because of gender. This result is received positively, because the CS Principles is intended to appeal to both men and women.

4.1.1 Positive Programming Experiences

Students indicated a "low confidence" score in their programming skills at the beginning of the semester. This was measured by students rating their overall programming experience on a 1-5 Likert scale. 88% indicated they had "no experience" or "little experience" in programming (σ=0.8); only 12% of students indicated "some experience," and only one student in each course indicated a moderate or high confidence in their programming skills. However, after taking the course, a majority of students moved from "no experience" to "some experience." The value of p is 0.015, which is significant at ≤ 0.05. Therefore, students in the variable group had a meaningful increase in their level of experience for programming after the course, which is a direct goal for the CS Principles framework.

4.1.2 Continuation of Learning CS

The one-year follow-up survey was used to measure whether students would pursue (or did pursue) computing after the course ended. A total of 25 students participated in the one-year post-survey (27%). Of the survey takers, 12 were female and 13 were male. Six students (24%) reported that they had taking additional CS or programming courses at the university in the past year. Without being asked, one student reported that they switched into a CS major specifically because the case study course helped motivated them to do so. Surprisingly, more students reported that they had taken online courses or website tutorials than formal courses at the university. Nine students (36%) reported that they had taken some kind of online learning to continue their pursuit of CS. Students had varying purposes for using the online resources, whether to learn a specific skill or to reach a specific objective, such as learning how to build a website, learning code to enhance their job skills or future career, or to help supplement their studies for other courses. Lastly, seven students (28%) reported that they continued their pursuit of CS by participating in hands-on learning activities during a workshop, summer internship, etc. This suggests that many students pursue their study of CS but not necessarily through a formal continuation in higher education, even if some did actively choose to change their major as a result of the course.

[7] The university for this case study uses a five-point grade lettering system, where an A=92-100%, B=83-91%, C=74-82%, D=65-73%, and F=0-64%.

4.2 Impact on Student Motivation

Our findings indicate that CS Principles positively impacted student motivation in learning computer science. CS Principles increased interest in CS as a field of study, helped students consider CS as a field of practice, facilitated positive programming experiences, and helped encourage students to continue learning CS after the course concluded.

4.2.1 CS Principles Improve Attitude about CS

Emergent categories and themes indicated that students had shifting attitudes about CS as a result of the course. The dominant category most frequently referred to in student open-ended questions were attitudes, knowledge, and definitions of CS. Analysis after coding indicated that 77.3% of students had a positive *attitude* towards CS (either a changed-to-positive or stayed-positive). 57.5% of students had a positive *definition* of CS. In addition, 47% of students had a *positive knowledge identifier* about CS—an example of a positive knowledge identifier would be "I like to build websites." Presumably this student would expect to be good at CS because of this interest and experience in web development. Nevertheless, 53% of students had a *negative knowledge identifier*, a student who assumes that one has to be good at hardware in order to be good at CS, leading to a negative knowledge identifier. There were nearly twice as many students who had a *positive change in definition* of CS than the students who had a *positive same*, or *negative* attitude about CS. Overall, the definition of "computer science" by students was often closely tied with specific knowledge identifiers and student perceptions and attitudes about their own ability and skill level in that knowledge.

4.2.2 Increased Interest in CS as a Field of Study

Students not only improved in their attitude about CS, their interests in CS as a field of study also increased. In the proxy pre-test and post-test survey, IMI survey items sought to measure whether the course increased their interest in CS. To determine if there was an increase in student interest in CS as a result of the course, a Student t-test was conducted between the pre-test and post-test measure of student interest used as part of the IMI scale. The two-tailed p=0.012, indicating a statistically significant difference. Qualitative findings also supported the quantitative data; for example, a female student reported that "*this course made me more interested in computer science and… start doing some computer science in my everyday life. My way of thinking has changed.*"

4.2.3 CS as a Creative Field of Practice

One of the key themes of CS Principles is to show students that CS is a creative practice. Students in CS Principles are also expected to demonstrate that creativity through practice. Survey questions sought to measure student's preconceptions about whether they considered CS as a creative practice and whether their exposure to the course changed this perception in some way.

During the pre-test, a majority of students thought that CS was a creative practice (55%), where 12% thought CS was not a creative practice, and 33% were unsure. After taking the class, almost no students were unsure about whether CS was a creative practice or not. The post-test showed a significant increase of students' positive perception, where 96% considered CS to be a creative practice. These data suggest that the content of the course helped shift student perceptions about CS as a creative field of practice. Another *t*-test was used, where p=2.52E-05, which is significant at ≤ 0.05, indicating a significant increase in students' perception of CS as a creative practice. A male student of color reported: *"I realized that my creative approach is actually beneficial to the sciences...it was a class that truly left a lasting impression on me."*

5. DISCUSSION

Findings show that CS Principles had a positive impact on students, indicating that the CS Principles curriculum framework supports student interest and motivation to learn introductory computing. In addition, CS Principles enabled students to consider CS as a creative field of practice, fostered more positive attitudes about CS as a field of practice, and increased the level of interest for students to pursue learning computer science in multifaceted ways. The findings from the follow-up survey also suggest that CS Principles established a strong foundation for students to pursue CS as a tool to support other academic interests. Although some students did change their major, most pursued non-traditional avenues to continue their learning. How they pursued their continuation of learning is nuanced yet persistent; to this end, the course accomplished the overarching goals of the framework. Therefore, CS Principles as a curricular framework for introductory students was observed to be a success in this case study.

5.1 Relevance of Self-Determination Theory

Generally, we found that Self-Determination Theory matched well with the overlying CS Principles framework. Lessons were also learned about the practical application of the subclass theories.

5.1.1 Self-Directed Autonomy

Students' attitudes toward the self-paced nature of the course were bimodal, either strongly positive or strongly negative. Students with positive views appeared to benefit from the self-directed learning, showed a higher degree of autonomy, and performed better (had higher final grades in the course). Students with negative views about the self-directed autonomy argued they needed more extrinsic motivation to keep them on track, such as having a hard deadline for assignments. Students also reported a dislike for the reading and reflection assignments the most, suggesting that the building of artifacts were viewed as more positive learning experiences. Although students reported the scaffolding mechanisms in assignments helped their learning, this suggests that too-much self-direction may not necessarily be beneficial to learning; some constraints can aid in creativity and productivity [2]. Therefore, removing course deadlines does not inherently support self-directed learning or autonomy. We recommend that practical applications of CS Principles incorporate "creative freedom with guidelines."

5.1.2 Constructing Mastery

Overall, constructionist elements in the course were found to enhance the CS Principles curriculum. The course structure also supported constructionism. Most of the students reported that their "best" experiences in the course happened when they were programming and building computational artifacts. Moreover, students appeared to experience aspects of mastery when overcoming challenges and solving difficult learning problems. Some students reported disliking the reading or "textbook" assignments, and commented that these assignments did not align with their goals and expectations for the course, again pointing toward constructionism as an appropriate model for facilitating introductory computing. Other qualitative data indicated constructionist methodologies allowed students the freedom to fail, and demonstrated the importance of feedback, which promoted learning. As one student noted: *"I liked the lack of tests a lot. I also liked how the emphasis of the quest grading was on effort because it made me feel more free to experiment with the coding and make mistakes."* Web-analytics from the course website also show that many students took advantage of redemption. Overall, constructionist elements of the course theoretically and practically matched CS Principles.

5.1.3 Connectivist Purpose

In terms of the more explicit technological connectivist portions of the course—specifically the social media features on the site—the students continuously utilized these options throughout the semester. For example, eight different forums were created by students, with over thirty-five topics in the forum, totaling 318 posts by students. In addition, seven student "groups" were created, averaging eight people per group. These interactions thereby support both the theoretical and practical application of the connectivist community purpose.

Students also reported a positive shift in their attitudes about CS, but more significantly, many students reported the importance and purpose that CS has on society. Most students viewed the course content as valuable and meaningful, beyond the context of the course itself. One student reported: *"This course has made me much more interested in and motivated to continue with CS. I feel like I already know so much more about technology in general, and how crucial it is to global development."*

5.2 Future Work

One limitation in this work is the methodological approach. Implementing the same curriculum within a high school course or additional undergraduate courses would confirm the replicability of CS Principles instantiating Self-Determination Theory. However, similarly to other CS Principles pilots, it is difficult to precisely replicate course curriculum when CS Principles is a broad framework rather. Studies that attempt to replicate previous pilots rather than reimagining them is a recommended next-step.

In addition, since all coursework and student interactions occurred on a website, replicating these efforts into a "distance learning" or online context is a practical extension to this work. Distance learning literature and research, whether it is Massively Online Open Courses (MOOCs) or an online undergraduate course, are often concerned about motivating and sustaining students to complete all of the online learning objectives [34, 1]. The core objectives of CS Principles, as well as the pedagogical flexibility offered by the framework, denote that this course could potentially scale to MOOCs and other distance-learning initiatives. We encourage research and teaching efforts focused on assessing self-directed creative constraints and scaffolding materials in distance learning within CS Principles. In addition, there needs to be more consideration in how to apply situated learning techniques into the context of an online course, particularly in the realm of constructionism and connectivism.

6. REFERENCES

[1] Ally, M. 2008. Foundations of educational theory for online learning. In T. Anderson (Ed.), *The Theory and Practice of Online Learning* (Fifth Edit.). AU Press, Athabasca University.

[2] Amabile, T. M. 1996. *Creativity in Context*. Boulder, CO: Westview Press.

[3] Arpaci-Dusseau, A., Astrachan, O., Barnett, D., Bauer, M., Carrell, M., Dovi, R., … Uche, C. 2013. Computer science principles: analysis of a proposed advanced placement course. In *Proceeding of the 44th ACM technical symposium on Computer science education (SIGCSE '13)* (pp. 251–256). Denver, CO: ACM New York.

[4] Ashcraft, C., Eger, E., & Friend, M. 2012. Girls in IT: The facts. *National Center for Women and Information Technology*. Boulder: University Press of Colorado. Retrieved from NCWIT: https://www.ncwit.org/sites/default/files/resources/girlsinit_thefacts_fullreport2012.pdf

[5] Astrachan, O., & Briggs, A. 2012. The CS principles project. *ACM Inroads*, *3*(June), 38–42.

[6] Behrens, A., Atorf, L., Schneider, D., & Aach, T. 2011. Key factors for freshmen education using MATLAB and LEGO mindstorms. In *ICIRA'11 Proceedings of the 4th international conference on Intelligent Robotics and Applications - Volume Part I* (pp. 553–562). Springer-Verlag Berlin, Heidelberg. Retrieved from http://link.springer.com/chapter/10.1007/978-3-642-25486-4_55

[7] Briggs, A., & Snyder, L. 2012. Computer Science Principles and the CS 10K Initiative. *ACM Inroads*, *3*(2), 30–31.

[8] Brown, T. 2008. Design thinking. *Harvard Business Review*, *86*(6), 84–92, 141. Retrieved from http://www.ncbi.nlm.nih.gov/pubmed/18605031

[9] College Board. 2014. *AP Computer Science Principles Draft Curriculum Framework*.

[10] Csikszentmihalyi, M. 1990. *Flow: The Psychology of Optimal Experience*. New York: Harper and Row.

[11] Deci, E. L., & Ryan, R. M. 1991. A motivational approach to self: Integration in personality. In R. Dienstbier (Ed.), *Nebraska symposium on motivation*, *38*(1), 237-288. Lincoln, NE: University of Nebraska Press.

[12] Downes, S. 2006. *An Introduction to Connective Knowledge*. Creative Commons License. Retrieved from http://www.downes.ca/post/33034

[13] Downes, S. 2012. *Connectivism and Connective Knowledge: Essays on Meaning and Learning Networks*. Creative Commons License. Retrieved from http://www.downes.ca/files/Connective_Knowledge-19May2012.pdf

[14] Duckworth, A. L., Peterson, C., Matthews, M. D., & Kelly, D. R. 2007. Grit: perseverance and passion for long-term goals. *Journal of Personality and Social Psychology*, *92*, 1087–1101.

[15] Dweck, C. 2006. *Mindset: The New Psychology of Success*. New York: Ballantine Books.

[16] Ericson, B., & Guzdial, M. 2014. Measuring demographics and performance in computer science education at a nationwide scale using AP CS data. *Proceedings of the 45th ACM Technical Symposium on Computer Science Education (SIGCSE '14)*, 217–222.

[17] Ericson, B., Guzdial, M., & Biggers, M. 2007. Improving secondary CS education: Progress and problems. In *SIGCSE '07 Proceedings of the 38th SIGCSE technical symposium on Computer science education* (pp. 298–301). New York, NY: ACM.

[18] Furst, M., Isbell, C., & Guzdial, M. 2007. Threads™: How to restructure a computer science curriculum for a flat world. In *SIGCSE '07 Proceedings of the 38th SIGCSE technical symposium on Computer science education* (pp. 420–424). New York: ACM.

[19] Kafai, Y. & Resnick, M., Eds. 1996. *Constructionism in Practice: Designing, Thinking, and Learning in a Digital World*. Mahwah, NJ: Lawrence Erlbaum Associates, Inc.

[20] Knowles, M. 1975. *Self-directed Learning: A Guide for Teachers and Learners.* New York: Association Press.

[21] Locke, E. a., & Latham, G. P. 2002. Building a practically useful theory of goal setting and task motivation: A 35-year odyssey. *American Psychologist*, *57*(9), 705–717. doi:10.1037//0003-066X.57.9.705

[22] MacLean, M. S., Mohr, M. M., Mohr, M. M., & National Writing Project (U.S.). 1999. *Teacher-researchers at work*. Berkeley, Calif: National Writing Project

[23] Malan, D. J., & Leitner, H. H. 2007. Scratch for budding computer scientists. In *SIGCSE '07: Proceedings of the 38th SIGCSE technical symposium on Computer science education* (Vol. 39, pp. 223–227). New York, NY: ACM. doi:10.1145/1227504.1227388

[24] Maloney, J. H., Peppler, K., Kafai, Y., Resnick, M., & Rusk, N. 2008. Programming by choice: urban youth learning programming with scratch. In *SIGCSE '08: Proceedings of the 39th SIGCSE technical symposium on Computer science education* (pp. 367–371). New York, NY: ACM.

[25] Margolis, J., & Fisher, A. 2002. *Unlocking the computer clubhouse: Women in computing*. Cambridge, MA: MIT Press.

[26] Mitgutsch, K. 2012. Learning through play - A delicate matter: Experience-based recursive learning computer games. In J. Fromme & A. Unger (Eds.), *Computer Games and New Media Cultures* (pp. 571–584). Dordrecht: Springer Netherlands. doi:10.1007/978-94-007-2777-9

[27] Owens, B. B., & Stephenson, C. 2011. *CSTA K – 12 Computer Science Standards*. New York, NY. Retrieved from http://dl.acm.org/citation.cfm?id=2325380

[28] Papert, S. 1980. *Mindstorms: Children, computers, and powerful ideas.* New York, NY: Basic Books.

[29] Papert, S. & Harel, I. 1991. *Constructionism*. Ablex Publishing Corporation.

[30] Rovai, A. P., Baker, J. D., & Ponton, M. K. 2014. *Social Science Research Design and Statistics: A Practitioner's Guide to Research Methods and IBM SPSS Analysis.* Chesapeake, VA: Watertree Press.

[31] Ryan, R. M. & Deci, E. L. 2000. Self-determination theory and the facilitation of intrinsic motivation, social

development, and well-being. *American Psychologist, 55*, 68–78.

[32] Ryan, R. M., & Powelson, C. L. 1991. Autonomy and relatedness as fundamental to motivation and education. *The Journal of Experimental Educational: Unraveling Motivation*, *60*(1), 49–66.

[33] Ryan, R. M., Connell, J. P., & Grolnick, W. S. 1993. When achievement is *not* intrinsically motivated: A theory of self-regulation in school. In A. K. Boggiano & T. S. Pittman (Eds.), *Achievement and Motivation: A Social-developmental Perspective.* New York: Cambridge University Press.

[34] Siemens, G. 2005. Connectivism: A learning theory for the digital age. *International Journal of Instructional Technology and Distance Learning*, *2*(1), 3–10.

[35] Snyder, L., Barnes, T., Garcia, D., Paul, J., & B. Simon 2012. The first five computer science principles pilots: Summary and comparisons. *ACM Inroads*, *3*(2), 54–57.

[36] Stake, R. E. 1995. *The Art of Case Study Research*. Thousand Oaks, CA: SAGE Publications.

[37] Tew, A. E., Fowler, C., & Guzdial, M. 2005. *Tracking an Innovation in Introductory CS Education from a Research University to a Two-Year College*. In SIGCSE '05: Proceedings of the 36th SIGCSE technical symposium on Computer science education (pp. 416–420).

[38] Utting, I., Cooper, S., & Kölling, M. 2010. Alice, Greenfoot, and Scratch – A discussion. *ACM Transactions on Computing Education (TOCE)*, *10*(4), 1–11. doi:10.1145/1868358.1868364.http

[39] Wagner, T. 2012. *Creating Innovators: The Making of Young People Who Will Change the World*. New York, NY: Scribner.

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