2016

Computer Science Education at The Claremont Colleges: The Building of an Intuition

Lauren Burke
Scripps College

Computer Science Education at The Claremont Colleges: The Building of an Intuition

By

Lauren Burke

SUBMITTED TO SCRIPPS COLLEGE IN PARTIAL FULFILLMENT
OF THE DEGREE OF BACHELOR OF ARTS

PROFESSOR DEEB
PROFESSOR PERINI

APRIL 22, 2016

Acknowledgements

I would like to thank Professor Deeb and Professor Perini for their support, motivation, and knowledge throughout this process. Both provided an integral role helping me when I ran to into trouble or had questions about my research or writing. I would like to thank the Claremont Colleges computer science professors who not only nurtured my interest and enjoyment of computer science, but were willing participants in this thesis.

I am grateful to my friends, Kinzie and Becca, and my family who were willing to read portions of my thesis and listen as I tried to plan out the chapters and generally talked about my thesis. I would like to especially thank Marina who was always willing to answer random questions about computer science.

Introduction

This work began with an investigation of computer science education at the Claremont Colleges. What becomes obvious, when discussing computer science at these schools is that in recent years teaching computer science has become a popular topic of discussion between educators, students and those within the technology industry as well as within popular culture.[1] What is often discussed in both the media and industry are the increasing number of jobs and the lack of diversity in both race and gender within the technology industry and the field of computer science. This has led to an explosion of literature on how computer science should be taught as well as literature and programs designed to increase both the general knowledge of computer science practices and minority involvement in the field. In relation to the Claremont Colleges, professors and the departments are aware of the current societal discussions of computer science and ideas on how to make the field more inviting to students from differing educational backgrounds. Computer science as a field of study has grown to encompass many different topics within the relatively short time it has been in existence. This has created an increase of interest in computer science and its potential applications are seen as wide ranging.

Throughout this thesis, I will discuss how the undergraduate computer scientist is trained, and how they learn what I am calling computational intuition. Computational intuition is a phrase that I am using to describe the methodology in which computer

[1] Often technology industry is referred to as the industry. These go beyond just the applications that are used on the phone, but also include items like pieces of hardware, say your cable box, the keyboard on your computer as well as many other hardware items that you might not think of as involving computer science.

scientists approach their problems and solve them through the use of computers. Computational intuition is described as a series of skills and a way of thinking or approaching problems that students learn throughout their education. This involves not only the specific skills and concepts, but the larger and more nebulous manner in which problem solving occurs within computer science. The main way that computational intuition is taught to students is through the experience they gain as they work on homework and classwork problems. To develop computational intuition, students learn explicit knowledge and techniques as well as knowledge that is tacit and harder to teach within the lectures of a classroom environment. Computational intuition includes concepts that professors and students discuss which include "computer science intuition," "computational thinking," general problem solving skills or heuristics, and trained judgement.

It is evident when talking to students and professors that there is the idea that students are supposed to develop what is called a "computer science intuition." However, the phrase "computer science intuition" does not in itself capture all of the meaning that the computer scientists have when they employ that phrase. Using the word intuition often points to an instinctive feeling or understanding and there is some element of this in computer science. There are gut feelings and snap judgments about how a computer scientist might solve or approach a problem. However, what is ignored within the concept of intuition is a sense of reasoning and the strategies that are employed to solve problems.[2] These snap judgments and ideas that make up the more common definitions of intuition and represent a single portion of computational intuition that the students are learning throughout their education.

Another facet of computational intuition is the idea of "computational thinking," which is a skill that students are expected to master. Computational thinking uses heuristic reasoning to deal with complex and open ended problems by using a process of logical analysis to break down the problem into smaller parts and then creating a serious of steps that solves the problems.[3] Computational thinking is one of the key components of how computer scientists solve problems and is a specific way of solving problems that is considered to be from computer science. Eventually, the smaller steps would be broken down into commands that the computer can performs to achieve the task asked of it. The ability to deploy computational intuition is demonstrated in the successful demonstration of solving a problem.

The variety of problem solving techniques within computer science, including computational thinking, can be described using the word heuristics. The idea of computational thinking is tied to the idea of heuristics because computational thinking uses "heuristic reasoning."[4] Heuristics is described as a "rule of thumb. It's a method of reasoning that is powerful and general, but not absolutely guaranteed to work."[5] The idea of teaching heuristics within scientific education is considered another important topic

[2] *Merriam Webster Online*, s.v. "Intuition."
[3] "Exploring Computational Thinking" Google For Education, accessed March 30, 2016 https://www.google.com/edu/resources/programs/exploring-computational-thinking/, paragraph 2.
[4] Jeannette M. Wing, "Computational Thinking," *Communications of the ACM* 49, no. 49 (2006): 34.
[5] Michael E. Martinez, "What Is Problem Solving?" *The Phi Delta Kappan* 79 no 8, (1998), 606.

closely tied to the idea of pedagogy. This is found both within the field of computer science as well as other scientific fields. Herbert Simon argues that:

> "In teaching problem solving, major emphasis need to be directed toward extracting, making explicit, and practicing problem-solving heuristics –both general heuristics, like means-ends analysis, and more specific heuristics, like applying the energy conservation principle in physics."[6]

Within computer science, there is a focus on teaching students heuristics, Chandhry et al, points to the importance of teaching students tools for problem solving during their first years of studying computer science. Throughout their research, they discuss the fact that answers and ways to solve problems are relatively unknown to the students, and as they develop as computer scientists they build more methods of solving problems and more knowledge.[7] What is more interesting is that this knowledge is not one which can be transferred easily; it is often something that can only be learned through practice, where the student is able "to generate the knowledge."[8] Professors can teach their students some hints of heuristics or give them examples on how to approach problems, but students must ultimately create their own strategies and ways of approaching problems. To learn how to solve problems students need to be given problems where they are able to figure out how to generate problem solving skills and techniques.

[6] Herbert A. Simon, *The Sciences of the Artificial* (Cambridge, Mass.: MIT Press, 1981)
[7] Nadeem Chaudhry, and Ghulam Rasool. "A Case Study on Improving Problem Solving Skills of Undergraduate Computer Science Students." *World Applied Sciences Journal* 20, no. 1 (2012): 34-39.
[8] Alexander Styhre. "Practice and Intuitive Thinking: The Situated Nature of Practical Work." *International Journal of Organizational Analysis* 19, no. 2 (2011): 109-26.

Heuristics do not always give the most accurate or even the most optimal answers, but they do allow for problems to be solved eventually. What is more important is they point to the situated nature of knowledge, as heuristics are a mixture which "involves an interaction of person's experience and demands of the task."[9]

"Trained judgement," is the last element of computational intuition and is borrowed from Lorrain Daston and Peter Galison. Lorraine Daston and Peter Galison demonstrate in their book, Objectivity, that the concept of trained judgment is part of the creation of the scientific self and how scientists approach their studies. In Objectivity, Daston and Galison focus on the use of images that the scientists create for atlases and the process they go through to represent that knowledge and make sense of it. This "trained judgement" is how scientists use their previous experience and gained knowledge to make informed interpretations of pictures.[10] This is demonstrated when computer scientists make choices about different algorithms which they could potentially use during their process of problem solving. There are many different algorithms that a computer scientist can choose from that might solve the problem, but the choice they make depends on the situation and the type of problem they are solving.[11] This demonstrates how trained judgment can be extended past the interpretation of photos and represents the manner in which computer scientists approach and solve their problems.

The building of experience within computer science performs a multitude of tasks. It helps students build their heuristics, including computational thinking, as well as

[9] Martinez, "What Is Problem Solving?" 606.
[10] Lorraine Daston and Peter Galison, *Objectivity*. (New York: Zone Press, 2010), 359.
[11] An algorithm is a procedure or formula, it is the step by step process for solving a problem

develops “trained judgment.” This training is important in aiding the novice to see and understand what they could not before.[12] It is the process in which they utilize their previous experience, and earlier solutions to problems to solve new problems. These processes are used to make decisions about how they are going to approach the problem presented to them.

I argue that the way the professors and students conceptualize, teach and socialize together are designed not only to teach specific skills and concepts within computer science, but help the students understand the larger manner in which problem solving occurs within computer science. The goal of the computer science education at the undergraduate level is to expose students to the different areas of computer science and prepare them for graduate school or working in the industry by building their computational intuition.

The Claremont Colleges

When discussing the undergraduate education at the Claremont Colleges, it is important to note how the computer science departments at the colleges are arranged as it greatly influences the students’ experiences at the colleges. There are five colleges that are considered part of the Claremont Colleges these are: Scripps College, Claremont McKenna College, Harvey Mudd College, Pomona College, and Pitzer College. Only three out of the five colleges have computer science departments; these are Harvey Mudd, Pomona, and Claremont McKenna. At these three colleges, there is mixing of both

[12] Daston and Galison, *Objectivity*, 359.

students and professors from all five schools; especially as the demand for lower level computer science classes continues to increase.[13] The main difference between the schools' computer science programs is in their introductory courses. Harvey Mudd College has its own computer science department with its own specific introductory sequence. Meanwhile, Pomona College and Claremont McKenna College share a department and have an introductory sequence separate from those taught at Harvey Mudd. These sequences cannot be mixed, meaning if a student starts an introductory course at Harvey Mudd, they must complete the next courses in the introductory sequence with the Harvey Mudd computer science department. These introductory sequences at the two different departments teach similar material, as the material is necessary to prepare the students for upper division classes. All departments have the goal of training their students to be as prepared as possible for either work or graduate school. However, there are differences in the number and organization of the classes and other items such as the programming languages used.[14] While students are tied to a specific colleges' sequence for the initial courses, the upper division courses do not have this requirement.

This means that within the Claremont Colleges, there are various ideas about how to train the next generation of computer scientists. Though differences exist, there are still many similarities in how these professors teach their students, what they hope their

[13] An examples of this is the Mudd introductory CS course being taught at CMC, as well as CMC and Mudd professors working on creating lower level computer science courses together.

[14] HMC uses python in their intro class, while CMC/PO uses Java. HMC has a three class introductory course, while CMC/PO has a two class introductory course. There is a lot of opinion on what languages is best for teaching students computer science and what levels are best.

students will gain from their classes and even the types of examples and homework problems the professors assign their students.

The Claremont Colleges is where I had my first introduction to computer science. Going into college, I never expected to take a computer science class. More importantly, I never thought that the subject would be interesting for me. In fact, when I did decide to take my first computer science class, I thought there was a strong possibility that I was going to be absolutely terrible at the subject. Taking classes within the computer science department allowed me to develop contacts within the departments with both professors and students. My computer science classes have been at Harvey Mudd College and Claremont McKenna College. I have not taken any classes Pomona College, though that is also indicative of the way in which the computer science departments with the consortium function.

At the time that I was determining my research topic, there were two driving forces that lead to my decision to study the computer science community at the Claremont Colleges. The first had to do with dynamics within the Scripps College campus, mainly with conversations that occurred over Facebook in the school year of 2014-2015. Within the Facebook posts on the Scripps College Facebook group and in ensuing conversations, you could quickly see that there was tension between the Scripps students who were science majors and humanities majors. This tension dealt with a number of issues and potential complaints for the Scripps administration. However, one thing was clear in the conversations. Though these were all Scripps students and they shared a common bond, there was a belief that specific differences existed between the two groups. These differences frequently lead to stereotyping of students as well class selection. Some students felt that science majors considered their science classes much harder than humanities classes leaving humanities majors feeling belittled. While on the other hand, humanities majors seemed to remain unsympathetic to science majors and how much time they professed to spend on their studies. It seems that on campus, students had begun to rank themselves and others based on majors as well as drawing lines between “science people” and “non-science people.”

The second aspect that was a driving force for choosing my topic was the many conversations that I had with other students about computer science after I had taken my first course. Many of my personal friends asked me if they should also try an introductory computer science course. My answer was almost always yes, because it is a science that is not often taught in high school and you might as well try it at least once. It is considered a valuable skill to have, even if it is only one class, and can make a person seem more employable. In fact, the latter is a reason that many people do take at least one computer science class at the Claremont Colleges. However, my friends would often argue that they were not “computer science people.”

After having these conversations, I began to wonder what exactly people meant by a “computer science person.” Especially, since this mirrors so many other statements such as people who say they are not a “math person” or are not a “science person.” I had begun to wonder exactly what those aspects were that made someone a computer science person and how that identity was born.

Methodology

This study was conducted over the course of the last four and a half months. During those months, I used participant observation in computer science classes at both Harvey Mudd College and Claremont McKenna College. Throughout this time, I conducted interviews with twenty-five Claremont College professors and students who had taken at least one computer science course. I had the opportunity to review homework assignments and grading guidelines that existed for some of the classes. Throughout these interviews, I discussed with students their experience taking classes at the Claremont Colleges and the skills that they thought were important to computer science and that they were learning throughout their education. I inquired about working on homework problems, and the methods in which they worked on their homework. When interviewing professors, I discussed what skills they thought were important for students to learn and the goals of their classes. Both students and professors were asked to define the field of computer science.

Literature Review

In The Structures of Scientific Revolutions, Thomas Kuhn discusses that revolutions in science are paradigm shifts that occur within groups of scientists who belong to a common field. Though, the concept of the paradigm shift in itself is not as important to my thesis, the idea of scientists as a group of people with a shared paradigm within which they operate is central. These paradigms include the fact that scientists have a shared set of beliefs that drive science and how a scientist performs their work. Drawing on the concept of a paradigm as a shared cultural belief, I take computer scientists and computer scientists-in-training to constitute a sub-culture with its own methods of transferring information to novices. This concept of scientists as a social group who can be studied is important, and as such can be studied with anthropological and other methodologies.

My study also builds on Bruno Latour's and Steve Woolgar's Science in Action, which argues that the creation of knowledge is a process through which scientists perform socially and can therefore be understood through anthropological methodologies. Latour states that this form of examining scientific knowledge and communities is important because there is often a disconnect between how science is said to be done and how it is actually done. Within my project, exploring the difference between what computer scientists do and how people believe computer science happens will be important to understanding how the identity of a computer scientist is both created and maintained.

The idea of scientists as a social group with their own norms and behaviors that are passed down is important to examining scientists and their practices within the field of anthropology. In Beam Times and Lifetimes, anthropologist Sharon Traweek, writes an ethnography on particle physicists at the Stanford Linear Accelerator and an accelerator in Japan. Traweek describes how the scientists go about their daily lives and interact with one another as scientists as well as the machines at their sites. The conceptual views of the machines that that scientists utilize in their research, and the way they interact with them, is as important as the way that the scientists interact with each other.[15] Traweek describes the physicists as a social group that reproduced through the "training of novices."[16] This training and education of the novice to a full-fledged physicist includes not only formal classroom training, but also tacit knowledge that is passed down from older physicists.[17] The transfer of knowledge, and the methods employed in this transfer, is important to shaping the novice physicists as they move forward in their careers. Traweek describes the potential novice's trajectory from undergraduate, to graduate and eventually the possibility of leading their own lab. Through this, Traweek addresses the most often told stories are male, and how the "cluster of characteristics is associated with success, a cluster that is part of our cultures construction of male gender."[18] This manner of considering the undergraduate scientists as a sort of novice who is being trained and having the transfer of knowledge is important, as well as the statements on the increasing complexity of knowledge that the novices are expected to gain over time.

As we move from projects that represent the study of scientists as a social group and the production of their knowledge, we examine the concepts of computer science education and studies on retention of students. Often this research on the retention of students is focused on the pedagogy and how classes within computer science are taught to encourage or discourage certain groups from continuing in computer science. As a topic, computer science education is becoming very popular, making the literature rather

[15] Traweek, Sharon. *Beam Times and Lifetime* (Cambridge, Massachusetts: Harvard University Press, 1992).
[16] Ibid., 74
[17] Ibid, 74
[18] Ibid., 105

expansive, especially focusing on the methodology of teaching computer science at the introductory level, and which practices help in keeping students within the field of computer science.

Turning to research performed by at universities, “Students Perceptions of Computer Science: A Retention Study Comparing Graduating Seniors vs CS Leavers.” This article examined a group of students at the Institute of Technology in Atlanta, Georgia who had all taken computer science classes. The students were then separated into two different groups: those who were graduating with computer science degrees, and those who left the computer science major.

They found that students left the major for variety of reasons including human interactions, rigor and workload, and connection of the skills they learned and application to the real world.[19] These trends are also what various computer science departments are investigating as they address the topic of how to increase retention rates (and diversity) at their universities.

Within the field of computer science education, there is vast and fast growing literature of different ideas and pedagogies that help retain people to the field of computer science, as well as beneficial actions that professors can take. Research such as that done by Lorie Carter, demonstrate how those who drop computer science classes often view it as stereotypically male dominated, as well as asocial. When examining successful practices of computer science, literature points to a wide range of methods ranging from

[19] Maureen Bigger, Anne Bauer, and Tuba Yilmaza. “Students perceptions of computer science: a retention study comparing graduating seniors with CS leavers”, *ACM SIGCSE Bulletin* 40, no 1 (2008): 402-406.

interactive approaches, to using game design as well as many other methods that might make students more successful.

Collins, Brown, and Newman present the approach of cognitive apprenticeship, which allows students to learn how to solve problems and apply knowledge that is transferred to them in new ways.[20] This is done through the interactions between their professors and other students. Cognitive apprenticeship is formed from the philosophy of constructivism, and focuses on the interactions that students have with their professors and peers as they work on problems or have problems modeled for them. A constructivist view of knowledge stresses that knowledge is embedded within a specific worldview, and underlines the importance of the interaction between the novice and more knowledgeable people within their field, as well as their own peers.

Though cognitive apprenticeship is based on the ideas of a constructivist understanding of knowledge, computer science does require a large amount of knowledge to be transferred to students. Within the cognitive apprenticeship model, there is an emphasis on having the student gain independence and discover the solutions to problems on their own. Because of this, the cognitive apprenticeship is the method in which the ideas and the necessary skills for students to develop computational intuition are taught. It is not the manner in which a specific concept such as a certain data structure or the syntax of a language is learned from a professor, these represent transfer of knowledge. It is during the process of problem solving and applying the knowledge

[20] A. Collins, J.S. Brown, and S.E. Newman. "Cognitive apprenticeship: Teaching the crafts of reading, writing, and mathematics." In *Knowing, learning, and instruction: Essays in honor of Robert Glaser*, edited by L.B. Resnick. (Hillsdale: Lawrence Erlbaum Associates, 1989), 453-494

transferred to them that the students are constructing new knowledge. This discovery of the link between the concepts and how to implement a solution are very squarely in the realm of constructionist theory by Piaget and Vygotsky.

Piaget's ideas of education allow for the students to bring prior knowledge to build understanding for themselves. Within this idea is the focus on how the learner reaches the conclusions or solves the problem.[21] Vygotsky, creating his theory of development around the same time, places an emphasis on social factors that help and contribute to learning.[22] Both ideals allow for the transfer of knowledge between a student and a teacher, but it is ultimately left to the student to apply this knowledge and solving problems; students are able to construct knowledge that is not explicitly stated to them.

To Do Computer Science

Chapters 1 and 2 focus on how the problem solving process in computer science is developed. These chapters focus on the introduction and teaching of heuristics to the students. Chapter 1 specifically discusses what computer science is and how professors and students conceptualize the field. The concept of what computer science is helps define the field within a practice of science, as well as point to specific aspects and beliefs about what the field is that computer scientists hold. These specific beliefs about what computer science is and how it is used, points to important aspects of the methodology of solving problems using computer science. How computer science is

[21] S. A. McLeod. "Lev Vygotsk." *Simply Psychology.*
[22] S. A. McLeod. "Jean Piaget." *Simply Psychology.*

defined provides the basis for why computational intuition is important and how it is used in solving the types of problems computer scientists are looking at.

Chapter 2 discusses the social interactions that the students have with their professors, students, and the tutors. Here is where I draw on the pedagogy of cognitive apprenticeship to understand, not only how specific skills and concepts are transferred to the students, but the role that social interactions take to help students understand the heuristics behind solving problems within computer science. By building an understanding of how students are taught problem solving, it will be easier to understand how the students are taught computational intuition.

Chapter 3 focuses on the homework assignments and how they are considered a primary area in which experience is gained. It is within homework assignments where computational intuition is used, applied, and developed by the students. It is here that “trained judgment” is the easiest to demonstrate and discuss within computer science, especially as this is where professors most often indicate how they expect their students to be able to use trained judgment.

Throughout these chapters, I will discuss the various aspects of computational intuition, and the practices throughout computer science education that teach the students these various characteristics that make up computational intuition. There are different ideas about how to teach computer science, ways of thinking, problem solving skills and methodologies that students learn throughout their time at the Claremont Colleges.

However, these are all taught in hope that students develop the computational intuition necessary for the computer science students to be successful later in life.

Chapter One:
What is Computer Science?

When discussing computer science and the formation of the computational intuition it is important to note how computer science professors and students define computer science. The way that these groups define computer science implies certain values that are considered important to the field of computer science, and what is taught throughout the classes. This in turn points to some of the main aspects of how computer science students are expected to develop their computational intuition and solve problems that are assigned in computer science courses.

While examining these definitions, there are two different viewpoints to consider, the outsider (someone who is not from the community) and the insider (someone who is part of the community). On one hand, there are stereotypical associations of what it means to do computer science. On the other, there is how computer scientists conceive the field of computer science and what it means to be a computer scientist. The way that professors and students define computers science also describes how students understand what they are studying and are building computational intuition.

The manner in which professors and students define computer science often goes beyond the short characterizations that are found in dictionaries. These definitions of computer science range from “the study of computers and their uses”[23] to “the science that deals with the theory and methods of processing information in digital computers, the

[23] *Merriam-Webster, s.v.* “Computer Science”, accessed November 11, 2015.

design of computer hardware and software, and the applications of computers."[24] In addition, there are also the stereotypes and what it means to be a computer scientist; however, these stereotypes often miss both what computer scientists do and how they go about their work. These stereotypes, and the way that computer scientists are often depicted in popular culture, frequently involve the notion that a computer scientist only sits in front of a computer and programs all day.[25] There is also the idea that computer scientists are unsociable and do not talk to anyone.[26] Beyond those definitions and stereotypes, it is the definition of computer science that the computer scientist gives that provides the basis of understanding some fundamental components of computer science across the various sub disciplines.

Research has shown that the way in which people view computer science is a contributing factor to why students choose to leave or stay in the field, or even decide whether or not to enter the field at all. In a study about the retention of students in computer science, Biggers et al, asked students what computer science was. They found that students who left were more likely to think that computer science was primarily coding, and often viewed the field as asocial, and is composed of repetitive tasks.[27] This perception came from taking only introductory courses that were programming heavy and did not necessarily introduce "bigger picture ideas about what computer science is."[28] It

[24] *Dictionary.com, s.v.* "Computer Science", accessed November 1, 2015.
[25] A program is a sequential set of instructions written by the computer scientist that allows them to tell the computer what they want it to do.
[26] Lorie Carter. "Why students with an apparent aptitude for computer science don't choose to major in computer science." *ACM SIGCSE,* date accessed December 10, 2015, *22.*
[27] Bigger, Brauer, and Yilmaz, "Students perceptions of computer science", 406.
[28] Ibid, 406

was because of these early experiences that students often believed that computer science was only coding and that meant many of the students lost interest in the subject.[29] These stereotypes described in the paper by Biggers et al, and the dictionary definitions are more of an outsider view of computer science that most people are exposed to. However, it is the insider view of the definition of computer science that is important to recognizing how computer science students understand what computer science is, how to approach their work and what computational intuition is.

The “Insiders”

When the professors and students are interviewed and are asked to describe or define computer science, they consider it to be an unwieldly task. This gave the impression that the answer could not necessarily be pinned down in a concise manner. In fact, one professor admitted “that [they] didn’t know if there was a good definition of computer science.”[30] Computer science is still a relatively new academic discipline with the first academic department in computer science arising post World War II. This was after many developments in computer science occurred in the areas of military research and technology.[31] As a discipline, computer science has come to consist of many sub-

[29] Bigger, Brauer, and Yilmaz, “Students perceptions of computer science”, 406.
[30] Professor Interview, Claremont Colleges
[31] Jeffery Shallit, “A Very Brief History of Computer Science”, *University of Waterloo,* 1995 , This page has a good brief history of computer science if needed. When discussing history of computer science, within the context of my research, it was most often discussed in the sense that computer science is a relatively young field. However, the history of computer science education in particular seems to be little written about, and the extent of how far back computer science and computing are understood tend to change depending on how far back and what exactly authors are considering to be computer science or computing. General historical consensus does point to the first academic computer science programs as showing up post WWII.

disciplines that extend to a wide range of topics.[32] These various sub-disciplines can range from “theoretical components, like algorithm design or even kind of going back and looking at various aspects of what’s physically going on with circuits and stuff… very broad and hard to define.”[33] On one hand, there might be computer scientists who are working on a more practical application of computer science such as software development like making apps for phones. On the other hand there are people who are working on more theoretical and mathematical theories of computer science or “you have people who say they are doing computer science, and are doing human computer interaction in a very abstract way.”[34] These people may not even be working with computers, but are still considered to be doing computer science.[35] However, the characteristics that students and professors used to define computer science are characteristics that can be seen across sub-disciplines. At its base, computer science is a field which is studying what computers do and what they can potentially do.

Professors draw attention to the connection that computer science has to other fields. They point to how these skills, which are learned in computer science classes, have become important to fields beyond computer science, and how computers can even perform tasks found in other fields. One professor describes the interactions between

[32] “Areas in Computer Science”, *Cornell University*, accessed August 15. 2015. This site gives examples of the different subfields within computer science. Explanation of some of the various subfields within computer science to be interrelated and separate at times, depending on the field. As undergraduates these interests in the various fields are what they are exposed to throughout their time at college. Often for graduate school, students choose a sub-discipline to study.
[33] Female, Student Interview, Claremont Colleges
[34] Professor Interview, Claremont Colleges. Human computer interaction is a sub-field of computer science which researches the design and use of computer technologies focusing on the ways that humans and computers interact with one another.
[35] Professor Interview, Claremont Colleges

computers and other fields by how "computers themselves, can exhibit creativity in the arts and sciences…. There are also interactions with other areas, psychology, and the arts… just about anything you can name really."[36] These interactions with other areas of interest include both the fields in science, technology, engineering and mathematics (STEM fields) as well as fields outside side of the STEM category such as economics, sports, and humanities. It is this fact that ties into some of the professors' opinions on why more students should take computer science classes. The interdisciplinary nature of the field is actually beneficial to both the computer scientist as well as the students. Non-computer science students and students of other science majors can help bring different understandings to computer science classes. One example was of an anthropology major that might bring interesting points about human behavior and user design that computer scientists might not think of.[37] These non-major students are also being exposed to skills that could help them in the future as the use of computers and the knowledge of computer science might become more important to these other fields.

Students often highlighted this diverse nature of the field as a main factor for their interest in computer science. Many students admit that they see computer science as a "tool to be applied to other things."[38] It is perceived as a way to innovate and create new and exciting technologies, and it is broad enough that a person can integrate their personal interests in it. For many of the non-majors, the very reasons for taking computer science courses is their ability to see connections by linking an understanding of

[36] Professor Interview, Claremont Colleges
[37] Professor Interview, Claremont Colleges
[38] Female Student Interview, Claremont Colleges.

computer science to their other areas of interest. Computer science and the knowledge of it is a tool to help these students realize their own interests and goals.

Computing

The word computing was one of the central concepts that both students and professors often considered fundamental to the definition of computer science. Computing is vaguely defined in dictionaries as "the use of a computer to process data and calculations" and the "act of calculating or reckoning."[39] It is the way in which computer scientists use computers to solve the problems they are presented with, often using mathematics, and create a sequence of steps that are known as algorithms. The term computing is one of the terms that are considered central to what computer science is. It is also one that can refer to a multitude of areas, as one of the professors pointed out; computing "can mean a lot of things."[40] This can involve discussing what the limits of computation are and looking at what is and is not possible. Even computer scientists working on areas that may not seem related to computation are arguably still thinking about it, and studying it.[41]

By defining one of the main aspects of computer science as computing, the term computing is intrinsically tied to the idea of "computational thinking." The term computational thinking is a specific type of heuristic and an element of the development of computational intuition in computer science. It is a method of approaching complicated and open ended problems using logical analysis to break down the problem

[39] *Dictionary.com, s.v*. "Computing", accessed November 1, 2015.
[40] Professor Interview, Claremont Colleges
[41] Professor interview, Claremont Colleges

in to smaller parts and then and creating a serious of steps that solves the problems. The ability to be a computational thinker is one that is very important for students to learn.[42] Thinking computationally is one part of the computational intuition or way in which computer scientists describe their problem solving endeavor:

> "How can you solve that using a computer? Recognizing the types of tactics you can use.... step by step. Because programs are followed logically in a step by step order and they do only exactly what you tell them to do, not what you intend for them to do. But what you actually program. And getting the actual step by step translating from how you would vaguely solve problem to how a computer is the computational thinking."[43]

This is especially important as computational thinking puts the focus on examining how does the computer scientists get the computer to do what they actually want it to do. There is both the process of developing an idea of how to solve the problem and then figuring out how to implement your solution in order to make the computer accomplish exactly what the computer scientist wants it to do.

This translation processes and the thinking that goes behind it, is part of the development of the computational intuition of the computer scientist. It is the ability to understand how to make the computer actually do what the computers scientists' goal is, and it is fundamental to computer science. Beside the general ideas of how one might solve problems and potentially creating an algorithm for it. The computer scientists use coding and programming to actually "talk to" the computers and give it the step by step instructions, to perform whatever the task might be.

[42] Professor Interview, Claremont colleges
[43] Student Interview, Claremont Colleges

What Is Coding?

Often many people think of computer science as creating a computer program or coding. However, not all computer science is programming. In fact, computer science is often considered more of the theoretical and abstract aspects of the theory behind computing. Programming or coding is considered the practical aspect of computer science and is necessary for the larger field of computer science; a person who writes these programs are often called a programmer. Programming and understanding how to write computer code is heavily focused on in the introductory sequences and is often a precursor to many of the more theoretical aspects of computer science.

Coding is stereotypically the most common word that comes to mind when discussing computer scientists and it also seems to be a word that those who are unfamiliar with computer science are unacquainted with. Code and programs are behind most of the components of today's products that include any type of electronics from computers, to hardware such as cable boxes, and the applications on phones. Effectively, a code is a "sequence of symbols…. that someone typed in, or copied, or pasted from elsewhere."[44] A program is a set of instructions, written in code, that tell the computer what to do and in what order.

To perform a specific task or solve a problem through computation requires a computer as well as someone who is able to program it. However, there are other aspects that are used to help create these programs. There are other items used within the

[44] Paul Ford, "What is Code?", *Bloomberg*, June 11, 2015.

computer such as the compiler (what makes the code run), an IDE (what the programmer write their code in), an interpreter or shell and the programming languages that the programmer uses.[45] All have effects on how the programmer writes the code and how that code may or may not work.[46] The choice of the programming language is even dependent on what the task is as some programming languages are better for specific tasks over others.[47]

It is up to the computer scientists to interact with the various components of the computer and programming to determine the manner in which they choose to represent the task so that it allows the computer to accomplish what they want. All the while, computers will "not leave wiggle room, and are totally unrelenting in their criticisms of you as a programmer."[48] This means the programmer must be precise in what they say and the programming language they choose determines how they will say or code it.

There are many different programming languages that one can use to create this set of instructions that the computer executes. The code varies and is based on the programming language chosen. Often the code can look unusual and makes no sense to someone who has not encountered it before, or there are some programming languages

[45] IDE or integrated development environment is an application where programmers can write their code in for software development. They often have source code editor, tools, and debuggers (what they use to get the bugs or problems out of the code). An Interpreter is for programming languages that takes the source code and translates it into something that that computer can understand and executes it line by line.
[46] A compiler and programming languages are both manmade objects that are used to interact with the computer. They are something that another computer scientist has created to interact with the computers. There are also other tools such as debuggers whose use is to help figure out what is wrong with the code, and possible sources of these errors.
[47] There are many different types of programming languages that a programmer can choose from. Instructions can be given in binary, assembly, or languages that look almost like English. Assembly is a lower level programming language that can tell the computer directly what to do.
[48] Professor Interview, Claremont Colleges.

that look similar to English. The compiler is the go between from the computer scientist and the computer. Programming languages need to be translated into lower-level instructions. This is the job of the compiler and the interpreter to take “the symbols you typed into a file and transforms them into lower-level instructions.”[49] It is only then that the computer can execute whatever it is that the computer programmer has written, and barring any errors or mistakes, it should work.

One aspect of computer science is knowing, or knowing how to find out, and think about all of these different aspects in coding, as this is the manner in which computer scientists learn how “to communicate things to computers,” so that tasks at the end of the day are performed or are communicated.[50]

What is Computer Science?

In this chapter I have revealed how computer science students and professors begin to understand the subject. Central to the concept of computer science is the idea of computing and computational thinking, a specific type of heuristic, as well as having the ability to work with computers and learning to communicate with them. The professor and the students point to a wide range of uses and subfields within the discipline of computer science. It is the fact that these students and professors see the many applications of computer science that makes it so difficult to create a succinct definition. They ultimately included in their definition the overarching concepts that are necessary in all of the sub-fields that exist in the field of computer science.

[49] Ford, “What is Code?”
[50] Student Interview, Claremont Colleges

This definition of computer science is important as I examine how professors and students interact to both transfer and build knowledge of this discipline in the classroom. The focus of these courses is so the students are able to learn the different parts of what computer science is and be proficient in the various aspects. As the students move through their careers at the Claremont Colleges, they interact with professors and other students, and learn more about what computer science is and gain experience to develop their computational intuition.

Chapter Two:
Apprenticeship in the Classroom

How professors define computer science, affects what the professors determine is important for the student to get out of their classes. What the students learn in the classroom, through doing their homework and even looking online, helps create a base of knowledge and experience through which the students can rely on as they move forward. The class problems and homework provide two different functions. The first is applying and testing the students understanding of the concepts that they have learned so far. The second purpose of the homework is for the students to gain the experience needed to develop their computational intuition. The building of computational intuition is attained with a way of learning called cognitive apprenticeship. The focus on this method of teaching is in the interactions that the students have with both professors and peers and allowing for social learning where the students to develop their computational intuition.

Cognitive apprenticeship presents a model where the student learns from the professors and their peers through a process of modeling skills and solutions and then trying to solve their own problems. Over time, students gain experience and are able to solve and discover the solutions on their own, and develop their computational intuition to understand how to do computer science. What is important about cognitive apprenticeship is the focus on the interactions between the students and others.

Taking from Vygotsky, learning is a social process, and it is the interactions between students, the professors and their peers that allow the student to learn. Furthermore, it is not just the classroom where students are learning, but also the process of working on homework assignments to gain experience by solving computer science problems using the computational intuition that they are building. The interactions between the professors, tutors, students and peers represent a framework of collaboration that allows the learners to create a better understanding than could be attained alone.[51] Furthermore, the actions of just listening to a lecture are often not enough to transfer knowledge to students even though lectures are often the main manner in which professors introduce new material to students. Lectures have a tendency to be the presentation and explanation of a unique piece of knowledge that can be applied in many different scenarios based on the problem at hand. An emphasis is placed on the homework as an important area for student to build their knowledge and the interactions between the young computer scientists and more advanced members of the fields are an important aspect of this.

The students gain skills through the introduction to the way that the groups think about their field of study and the development of their relation to others within their field. This transfer of knowledge is important and similar to other sciences; young computer scientists need to be taught how to work in their field. In examining the transfer of this knowledge in physics, Sharon Traweek discusses how knowledge does not only include "formal education, but also in the daily routines…"[52] Similar to how physics students learn their understanding of their field, as taught through conducting and observing controlled "experiments." Computer science is taught through participation in controlled

[51] J. G. Greeno, A.M. Collins, and L. B. Resnick. "Cognition and learning." In *Handbook of Educational Psychology*, edited by D.C. Berliner and R.C. Calfee. (New York: Macmillan, 1996), 15-46.
[52] Traweek, *Beamtimes and Lifetimes* 76.

homework assignments. These are important because they are problems that the professors know the solutions or possible solutions to. This allows the students to know if they are applying their knowledge correctly and if their heuristics and trained judgment lead to working answers. It is during these computer science classes and homework assignments where the skills students learn are applied in different ways and they are able to discover how to solve problems.[53]

To understand how the students learn computational intuition it is important to examine the knowledge and experience they gain through interactions that they participate in. The transfer of knowledge itself occurs through the process of people interacting with one another by sharing ideas and working together on problems that are assigned. There are many different ways these relationships between one another are developed and encouraged, such as informally working with other students, talking to professors or tutors and even requiring some homework to be worked on with a partner. Working with a partner within computer science is common and is encouraged as often one student or developer might have different knowledge or strengths than their partner, as well as help generate ideas or think of solutions that a person might not produce on their own, and catch mistakes that might be made. Computer science is often thought of as a highly individual subject where the lone programmer works through problems and projects that are assigned to them.[54] Though many of the students say they enjoy working by themselves more than with partners, there is still a high level of interaction between

[53] Traweek, *Beamtimes and Lifetimes,* 76
[54] Carter, “Why students with an apparent aptitude for computer science don't choose to major in computer science.”, 30

the students, tutors, and professors as they work on their homework or projects throughout their schooling. The colleges provide help in the form of tutors which are referred to by different names at each of the Claremont Colleges such as: tutors, grutors, or mentors; however, throughout this chapter I will be referring to all of them as tutors.[55]

These various interactions within the educational activity and the inherently social process engages the students and they in turn utilize their past knowledge and the experience gained in these interactions to help them develop their computational intuition. The more interaction the students have with people within their field, the more they are able to gain information and knowledge to solve problems. Professors often utilize different techniques, combining lectures and homework assignments to transfer knowledge and promote the development of the computational intuition in the students. They often start with the explanation of the subject matter, and how to use the skill or concept they are teaching, and then move the students to eventually produce their own work (homework or projects) to demonstrate their knowledge and build experience. What I shall next be discussing is what a normal day of class within a lower level computer science classroom might look like.

[55] Each school uses a different terminology for what they call the other students who help peers on homework in classes they have already taken. CMC calls their students tutors, PO mentors, and HMC grutors. Grutors stands for grading tutor as the students are often the ones who are both grading and providing the tutoring help. Most schools use the tutors to grade homework and provide extra help to the students, though importantly tutors do not grade exams that are given throughout the classes.

The Classroom

At the beginning of the class, which in this case is the second course in the introductory computer science sequence, students make their way into the room and find their seats; they chat with each other discussing everything from gossip to the homework they are working on for this class. Often the students take their computers out and go online while waiting for the professors to come into the room.[56] The professor walks into the classroom and begins to set up the lecture slides for class that day. A few minutes into the official start of class, the professor begins the lecture. The professor talks for a bit, discussing important data structures that the students need to know and then the professor calls on students to have them recall what the topics of the last classes were. This part of the lecture is mostly dominated by the professor, as they discuss the topic of today’s classes. In front of me, the various students flit between different internet webpages, doing everything from looking at the homework and lecture notes, to talking to friends on Facebook, and even online shopping. These habits are maintained throughout the lecture, and are probably why some of the computer science professors ban the use of laptops in their classrooms.

Today the topic is Binary Search Trees (BST), which is a data structure within computer science.[57] The students being taught the various elements of a binary search tree, such as how the data is stored and the concepts and thought processes that are

[56] Interestingly, not all CS classes allow computers to be brought into the classrooms. Some professors find this distracting to learning the material in classes.

[57] BST is a data structure that stores things in a form of the tree, making it easy to search for specific data within the tree. The tree as a root and the internal nodes store a key. Each key has two sub-roots which are often denoted as the left and right subtree, and the key is always greater than the left sub-root. The information in a BST is often denoted by a node which is a single element in the tree.

behind the creation of the BST. One aspect of the BST is the traversal of the tree, which is the way in which the program would print the different aspects of the tree. While describing this process, the professor asks the students to anticipate what number would be printed next. Throughout the class, the students ask clarifying questions about what the professor is presenting, but the class continually moves forward. After explaining the main aspects of the data structure, the students watch as the professor turns to them and asks them to demonstrate their knowledge of what they just learned.

The professor has the students work in small groups and asks them to figure out how to create a function that would delete a specific element in a tree would work within the BST. The students turn to their neighbors and a low murmur of voices fill the room as the students discuss the answer to the question. The professor waits patiently at the front of the room for about five minutes before calling the attention of the class. The professor calls on a student and has them answer a question, commenting on the student's ideas and allowing other students to propose their ideas.[58] After the question has been satisfactorily answered, the professor continues the lecture by alternating between giving information to the students, having them work in small groups, and answer questions. The professor tries to have all of the students answer a question, and creates an environment where students need to recall previous information and apply it to what they are learning now.

This is what the typical day is like in a computer science classroom. The class periods often follow the same sort of interactions between the professors and the students

[58] Professors utilize many different methods of class participation. One class uses a notecard method so every student is called on at least once in a class. Other professors wait for students to raise hands, and some just call randomly.

in the classes. However, the manner in which these interactions exist in the classroom is often dependent on the individual practices the professors establish in their classrooms and interaction with their students. There are parts of the class that are lecture to provide specific knowledge to the students, while other parts allow for the students to demonstrate that they are actively listening. These classroom experiences are one of the key means for the students to start gaining skills and understanding different aspects of computer science that they will later employ as they work on their homework.

The interaction between professors and students mostly occurs in two different forms. The first, and most common, is during the class period when the professors are teaching their classes. The second area of professor-student interaction is during office hours, which are hours outside of class where professors guarantee that they will be in their office for students to come ask questions. However, many students point out that they mainly talk to their professors during the class periods and rely on alternate help during homework hours such as the use of tutors. Alternatively, there are other opportunities and places to find helpful information that the students might require within the educational structure of the computer science programs.

So Where Are the Textbooks?

An important distinction between computer science at the Claremont Colleges and other sciences is the lack of consistent use of textbooks within computer science classes. Commonly at the undergraduate level, the sciences will have students learning out of a textbook; it is the way that "each new scientific generation learns to practice its trade."[59] However, at the Claremont Colleges, textbooks are often not used. Professors might sometimes recommend them to the students as an additional resource; however, for the most part students do not use them. This is different from many other fields of science where these textbooks often represent an accumulation of knowledge within the subject that students "mainly rely on" until they are working on their own research in graduate school.[60]

Sharon Traweek points to how within physics textbooks students are not only learning knowledge they are learning that the "interpretation of physics is not to be challenged."[61] There are the stories that these students are learning about the "heroes of science, and their own limited capacities."[62] The use of textbooks is not only that which teaches the students, but tells them the stories of how and who has created modern physics.

Yet, computer science at the Claremont Colleges does not have these textbooks, though often students might learn of famous computer scientists or about who created which programming languages. The books that the professors do recommend are often primers and filled with the rules of a specific programing language. The explanation for why few of these textbooks are used is found in two differing reasons. The first is that the professors provide most of the relevant information that is necessary for the students. The

[59] Kuhn, *The Structures of Scientific Revolution*. (Chicago: University of Chicago Press, 2012.), 1
[60] Ibid, 164
[61] Traweek, *Beamtimes and Lifetimes,* 75
[62] Ibid, 75.

second fact is that much of the information that students would be looking for in textbooks is easily available online. This means that the classroom, the professors, their peers and the internet is often the student's source for gaining their knowledge.

The Professors

The classroom provides the opportunity for a professor to share with their students a specific piece of knowledge. This is done through the process of introducing students to the concept, demonstrating the concept with examples, then seeing if the students can apply what they have learned to another problem, often demonstrated through the homework assignment. Though there is a large amount of time spent with the professor talking about aspects of what the students are learning, the professors provide opportunities to students to discuss with each other or work together both inside and outside of class. Students learn concepts and how to apply these concepts by working through examples or problems that are given to them by their professors. These problems are designed to demonstrate a student's understanding of the knowledge. Throughout this process, students find themselves interacting with many different people to find the answers.

These processes through which the professors facilitate a student's learning, especially throughout the process of working on homework, are similar to the constructivist pedagogy of cognitive apprenticeship. Within this relationship there are the interactions between the professors, the tutors, and the students which lead the students to answers or learning how to problem solve within computer science. The focus on cognitive apprenticeship is not only one in which the professors share knowledge but also the “heuristics (or know how) vital to using such knowledge in practice.”[63] Computer science is often defined as a subject that values “problem-solving” and considers the ability of its students to problem-solve using computers as one of the most important aspects gained through learning. More often than not, the students learn skills that they are expected to apply throughout their education and later in their careers in computer science. The concept of being able to apply the knowledge is especially important because it is during the process of applying the knowledge that the students recognize that they are learning new information for themselves as they solve problems through their interactions with others.

The homework assignments are where students can apply what they learned and are, for the most part, the interactions that occur with peers, tutors, and professors about homework are the focus of the cognitive apprenticeship. The ability to teach skills and concepts in relation to actual situations in which they are applied is an important aspect to the success of a cognitive apprenticeship models.[64] As the students move through their education, often the homework assignments and projects that are assigned become more complex. It is during homework, that students are able to apply the skills and concepts they have learned in class to more real-world contexts and situations.

The first aspect of a cognitive apprenticeship occurs as the professor, within the classroom, explains a piece of knowledge to the students and perhaps models it or works

[63] Gergen, Kenneth J., *Relational Being*. (Cary: Oxford University Press, USA, 2009), 251
[64] J.S. Brown, A Collins. And A Duguid. “Situated cognition and the culture of learning” *Educational Researcher* 18, no 1 (1989): 32.

through a problem with the students. Throughout this process, the professor will explicitly state or write down the steps that they are taking to show the students. [65] Using the earlier example of what occurs in the classroom, this modeling is represented by the professor describing the different aspects of the BST which is more informative than modeling. However, after explaining the structure of the BST, the professor will walk the students through a problem using the structure of the BST, so that they have an example of how to solve a problem.

From there, the professor may have students work on their own problems, in the class with their neighbor, or later when the students are working on their homework. Here is where the students demonstrate their ability to understand the heuristics or problem solving skills, combined with the concepts that they have learned in the classes. During these times the professors are available during office hours to help the students work through any problems or difficulties that they might be having with the new knowledge. This is an important aspect of the cognitive apprentice, where the students are applying the knowledge they have learned under someone who is considered an expert.[66] This makes these interactions with the professor's crucial moments, where the professor can gauge the student's understanding and where they need help. Though, depending on the days, homework schedules, and if there is a test coming up, these office hours might be more or less busy, affecting whether these interactions are one-on-one or a small group.

[65] Collins, Brown, and Newman, 482
[66] Ibid, 463

Often on the day or so before a homework assignment is due or a test is scheduled, there can be quite a lot of people to be found crowding into a professors' office for help.

It is during these office hours and tutoring hours that the professors engage in coaching the students. This means the professors are giving the students necessary help, feedback or hints to solving the problems, or perhaps demonstrating methodologies in how to solve or approach the problems.[67] Students will come to these office hours with questions about concepts or their code. These questions could be anything from discussing the logic of how to solve a problem or finding a syntax error and helping to debug code when everything goes wrong. It is within this vision of coaching that the social nature of both computer science and learning is demonstrated. The professors are not the only ones who are providing these functions and there are others within the educational support system that are important as experts and provide help. These are the student tutors who provide help for other students who are taking classes that they have already taken. The professors and the tutors will not give the students the answers to their questions, but will instead try to guide the students towards how to solve the problem or clarify confusion. These hints and instructions both allow a method for a student to continue to try and problem solve on their own, while also providing any necessary support for the students. Eventually the student will be able to solve their problems on their own, and will not require as much help from the professors or the tutors.

[67] Collins, Brown, and Newman, 463

Tutoring from other students is considered by many of the students, one of the key means in which they interact with another person to solve problems. Tutors are part of the education support system built into the various computer science departments at the Claremont Colleges and are students who have already taken and mastered the class. They are considered an extension of the professors and work in close direction with the professors. In fact, students at the Claremont Colleges will often interact with the provided tutors to a greater degree than going to professor's office hours to seek their help.

The next methods of cognitive apprenticeship are articulation and reflection. This is where the students get to demonstrate their knowledge, often through homework or tests, and are able to compare their solutions or discuss with the professors or tutors.[68] As the students move through their education, the problems that are given to them increase in difficulty. This allows the students to both to articulate what they have learned as well as demonstrate how much of the thinking process they have absorbed. Furthermore, during the process of tutoring and working with the students, the tutors and professors force the students to think out loud, which both allows the students to think through the logic of the problem and often solve the problem they are having.[69]

[68] Anne K. Bedner, et al. "Theory Into Practice: How Do We Link?" In *Constructivism and the Technology of Instruction: A Conversation*, edited by Thomas M. Duffy and David H. Jonassen. (Hillsdale, NJ: Lawrence Erlbaum Associates, Publishers, 1992), 28.
[69] Collins, Brown, and Newman, 482.

However, tutors and professors are not the only means in which the students are able to get help and information. There are also interactions between the students, their own peers, and the internet that help with learning and seeking out computer science knowledge.

Skill Development Outside the Classroom

One important aspect of computer science is that the professors, tutors and the scaffolding creating by the cognitive apprenticeship model are not the only sources of knowledge and information. However, these interactions maintain the importance of social interactions and the construction of knowledge by an active learner. In fact, students are encouraged to use other sources that they can go to for help or to ask questions. Students often find themselves working with other students who are in the same class; these partnerships can be formal or informal working relationships. The final source of information is the internet, though there are strict requirements on what a student can or cannot search for. Students are allowed to search for information such as an error message to figure out what it means or possible solutions or specific syntax that are associated with it. Students are not supposed to search for information such as the solution for specific problems they are given. To not follow these rules as set out is counterintuitive to the classroom goals of having students be able to solve problems themselves, as students will not learn everything. Furthermore, to take code form a source or look up solutions is something similar to committing plagiarism. Different sources of information are used in addition to the professors and tutors.

Outside of the classroom, students often find themselves working with or interacting with their own classmates in a collaborative learning environment. These collaborative learning environments are important to the constructivist way of learning. These interactions help the students work towards learning what they need to help them solve their homework problems. The amount of interaction between one student and other students can vary depending on what classes they are taking and the students own personality; however, there is quite a bit of interaction between the students. These interactions can involve trading jokes that are computer science related or working through homework problems that students might be having trouble with. Though there is a community as the students hangout outside of class, there is the importance of interactions while working on homework. Often students might discuss certain issues or ideas and methods of approaching a problem in more informal ways than what is necessarily called “pair-programming” which is addressed later. They often will discuss certain coding problems as well as possible solutions to these problems. These are still collaborative endeavors that allow for a level of knowledge transfer between students. Gergen discusses how collaborative writing can help a student by offering “complementary information, perspectives, and opinions, thus teaching each other and contributing to a richer and more informed product.” [70] The goal of working together is to expose the students to different viewpoints or ways of approaching the problem, through

[70] Gergen, *Relational Being,* 260

this, the students “try and understand the alternative view.”[71] Encouraging the students to work together, whether it be informally or formally, allows for students to share the knowledge or ideas that other students may have. This is the same process that occurs in the form of collaborative programming called pair-programming.

Students may interact with other students in a practice called pair-programming which can be a formal or informal process. Professors may make it a mandatory or optional. This process of formally working with other students is a situation created where the students work together and solve problems as a group. Pair-programming is a process where there are two programmers working together on one computer. The person who types on the computer and writes code is called the driver and the other person who reviews the code as it is being typed is the navigator. The two students in the pair work on the entirety of the assignment together, thinking through the logic of the problem and discussing approaches to solve the problem. As the students go through the homework, they switch off on the roles of the driver and the navigator every thirty minutes, allowing both students to participate in both roles.

This method of working on computer science programs is often encouraged or mandated because of the knowledge transfer benefits that are seen through the process of it. This is especially true in a situation where one student might hold more knowledge

[71] Thomas M. Duffy and David H. Jonassen, introduction to *Constructivism and the Technology of Instruction*, ed. Thomas M. Duffy and David H. Jonassen, (Hillsdale: Lawrence Erlbaum Associates, Publishers, 1992), 27

than the other student and is able to answer questions about a particular topic.[72] Though within the class setting many of the students have fairly similar knowledge levels, they often do not have identical knowledge, "a certain amount of knowledge transfer takes place in all PP [pair-programming] sessions."[73]

Pair-programming, whether it is formal or informal, helps build a collaborative environment which allows for social learning and multiple perspectives. Where there is a goal of sharing arguments and evaluating the evidence for them. Pair-programing forces both of the students to discuss and think about the ways to solve the problem they are given which is helpful because the two students may have two slightly different methods of approaching the solution of the problem. Throughout this process, students learn from one another, and evaluate the reasoning behind what their partner is saying. Often, there are multiple ways to solve a problem, so together they negotiate the various methods and ideas. They may or may not agree, but each is using the knowledge they have gained to help make these decisions. In addition, one computer scientist may know more than the other and is able to help make a better decision and in the process teaches the other computer scientist a new way to solve the problem. As the students discuss how to solve the differing solutions, they weigh the pros and cons of the different methods of solving

[72] Laura Plonka,Helen Sharp, Janet van der Linden, Janet, and Yvonne Dittrich. "Knowledge transfer in pair programming: An in-depth analysis." *International Journal of Human- Computer Studies* 73 (2015), 66 In a situation with a paired-programming dynamic of an expert and a novice, working together often takes on aspects of cognitive apprenticeship. As the expert works with the novice they often perform the task of modeling, or verbalizing thoughts, and coaching while observing the novice work. This demonstrates how the cognitive apprenticeship model is how novice computer scientists continue to develop their skills even after they graduate from college.

[73] Ibid, 77

the problems, which in turn develops a student's heuristics methods and add experience to their trained judgment.

From there, the computer scientists might start the actually coding of the program. Here is where the navigator can catch syntax and other errors, and discussions of how to actually code the program comes into question. This also includes the process of debugging the code or being able to find the areas where the problems in the code are arising. It is during both the debugging phase and writing the program phase where students may start using a search of the internet for help. Internet searches are often used for help with syntax and whether or not a specific language might have a built in function that could be helpful or perhaps even decoding what error messages that a student might be getting.[74] Google and communities like Stack Overflow represent sources of information where one can find help on their piece of code, or perhaps find similar problems to the ones that a student might currently be having.[75] Internet searches and the process of being able to find accurate answers from a site is considered one of the most important skills and represents access to a wider computer science community beyond the one that exists at the schools. Professors state this skill as important and sometimes have students perform exercises where the students are tasked with finding information online or practicing the best methods of doing Google searches. This often involves taking something like error messages that a student might be getting and searching for it, and

[74] Functions are something that executes a task. A built in function will combine the instructions to do the task in a single line of code.
[75] Stack overflow is a websites were people with specific problems are able to ask their questions submit code, receive answers and review of their code, as well as answer questions that occur on the website. This can allow a student who has a question to see if there are similarities, if the solution might be similar as well as a wide range of answers within the comments section.

showing that the student can find helpful articles which might solve the problem. Professors might give examples of helpful searches or what websites might hold the best answers.

Computer science students are given a wide range of tools that they are able to use outside of the classroom to find answers and knowledge they might be looking for. Throughout their time of socializing with other students and even other computer scientists on the internet, they are benefitting and learning skills or new problem solving approaches they might not have thought of before as they work on their homework. The sociality of computer science is one of the main ways in which students learns. There is of course, the one glaring aspect missing from these ways of students getting information, textbooks, especially since they are so common in other scientific subjects.

All to Develop Computational Intuition

I have examined how knowledge is transferred between the professor, the tutors, other students and the internet, through the interactions and relations that the students are building throughout their time during and outside classes. Often the professors' lectures are one of the main methods in which concepts may get introduced to students, especially as textbooks are not that commonly used within the Claremont Colleges. Computer science is often taught to the students using a methodology of cognitive apprenticeship, where the students are interacting with the professors to help build knowledge and experience solving problems. One important aspect is the multitude of people that students can interact with to find help or discuss problems. The students throughout the class are taught concepts from the professors, trying to applying the knowledge they have learned, and working with members of the community to solve problems.

This cognitive apprenticeship and social learning helps create and develop the computational intuition that the students need. This is a certain way of thinking and approaching the problems within computer science, using the tools and concepts that the students have learned from class, each other, and the internet. The professors and the students both agree that computer science requires a “different way of thinking,” which is ultimately the manner in which the students are learning throughout their classes and experiences. Within computer science, homework is used for two different aspects. This chapter has shown how homework works within the idea of cognitive apprenticeship and social learning. In the next chapter, I will continue to concentrate on homework and focus more on how homework assignments are one of the main areas in which students build the experience necessary for computational intuition, specifically good heuristic reasoning and trained judgment.

Chapter 3:
Homework

Though there is much information, knowledge, and the development of computational intuition that occurs within the interactions of the students with their peers and their professors, homework is the primary means in which the students build their experience when they are working on various projects using computer science both inside and outside of class. Computer science students are working with their peers, class tutors, and professors on homework, as well as participating in hackathons, and other computer science related activities; during which the students are developing the necessary skills, or computational intuition.[76] The professors hope to develop computational intuition in their students so that they grow as computer scientists and learn how to both approach problems and increase their understanding of computer science. It is through the computer science education that the students gain an understanding of computational intuition, both the heuristics and the trained judgment, and are later able to make judgments about choices about their approach to problems that are given to them. Throughout the computer science student's education, the ability of the students to apply heuristics and trained judgment occur in different ways, but homework is maintained as a key area in which students develop as well as demonstrate their skills in both areas.

There are many different methods for developing computational intuition in computer scientists. This can involve working on activities like hackathons, summer

[76] A hackathon is an event that computer programmers and software developers collaborate in software projects, normally over an intensive time during an event. Often there may be a presentation where the different groups present on the item that they put together. At the Claremont Colleges the hackathon is put on by students for other students and normally occur once a semester.

internships where they are expect to perform specific tasks, as well as working on problems of their own. The primary method in which computational intuition is developed within the education of the computer scientists at the Claremont Colleges is through the homework problems that are assigned to the students. Homework assignments are the manner in which students are able to demonstrate their knowledge in a context that might be considered more practical, and the situated work is something similar to what they might see later in life.[77] This is especially important to the idea of cognitive apprenticeship that was mentioned earlier, because the use of social interaction is central to the transfer of skills, knowledge, and problem solving tactics. The students need to be given homework that moves them past what is considered to be more abstract ideas.[78]

How to "Do" Computer Science

Chapter One's discussion on what computer science is shows us that it is chiefly a problem solving endeavor. So it comes as no surprise that homework assignments within computer science normally involve aspects of problem-solving. This ability to problem solve is one in which the students think logically, utilizing computational intuition. Different classes have different types of homework, which help train the eye and mind of the computer scientist. Homework problems for a class such as Algorithms are going to be different than homework for a class like Data Structures. At the end of the day these

[77] This is part of a learning theory called situated learning by L. Lave. This is the idea that learning needs to be situated in the context and culture that it would occur. Like many other theories from the constructivist philosophy, there is a focus on the community and interactions between other people within the community.
[78] Brown, Collins, and Duguid. "Situated cognition and the culture of learning" 32-42.

classes all demonstrate a problem solving endeavor that eventually can be implemented with a computer.

While students practice their problem solving on these homework assignments, they are acquiring experience with new heuristics and knowledge necessary to solve a problem. Within this, there are heuristics that the students learn and develop on their own and others that computer science professors will teach them. Techniques such as the ability to reduce one problem to another and seeing the relationships between one problem and other problems that students have already seen and solved are part of the heuristically reasoning that is often taught in classes. It is considered critical to be able to recognize similarities and reuse existing knowledge and ideas. The hope is that at the end of their classes students will understand how to pull this existing knowledge from previous experiences together to solve problems and be able to reduce larger problems to less complex and familiar ones.[79]

Being able to connect a problem that the students are currently working on with those that are closely related is a concept also developed in other fields that are focused on problems solving and where there is a focus on teaching students heuristics. Within mathematics, one of the ways Polya argues that problem solving occurs is to "find the connection between the data and unknown."[80] This is the thought that most problems are connected to other related problems that someone might already know how to solve. In undergraduate physics training, the same idea is applied where students are shown "how

[79] Professor Interview, Claremont College
[80] George Polya, *how to Solve It*, (Princeton: Princeton University Press, 1957), XVI

to recognize that a new problem is like this or that familiar problem."[81] Though within computer science the concept of "reducing a larger problem" is where the students might take a larger more complicated problem, and see that it might be made up of smaller more familiar problems. Often problems "defy one-shot solutions; they must be broken down."[82] Breaking down problems, or computational thinking, is one of the heuristic methods through which problems are often solved.

During the homework and problem solving process, there is the prospect that mistakes are going to be made. These "mistakes made along the way must be accepted as inextricable from the problem-solving process."[83] Since heuristics does not necessarily give the correct answer all of the time; the students are learning how to adapt to achieve their goal. This gives a high likelihood, that along the way students are going to make mistakes as they are solving problems. A point that was often highlighted in conversation with both professors and students was the notion of being comfortable with making mistakes. This was a trait that was especially discussed by professors, who said that "it is helpful to be comfortable with experimenting and sort of making mistakes and just sort of exploring and just being curious and all those sort of liberal artsy kind of things."[84] This curiosity and willingness to make errors points to the manner in which students can experience and build the computational intuition that they must learn.

[81] Traweek. *Beam Times and Lifetime,* , 77
[82] Martinez, "What Is Problem Solving?", 606.
[83] Ibid, 607
[84] Professor Interviews, Claremont Colleges

Making mistakes and being okay with failure is especially important as a great deal of computer science is filled with parts of programs that do not work correctly and therefore need debugging. Students often discussed how the ability to work through frustration and not give up on a problem is intrinsic to being successful in computer science. At times, computer science tasks can become tedious. Being able to work with the inherent frustration and the attitude a student takes towards it, can affect how much the student may or may not like computer science.

This, of course, is not a comprehensive or exhausted list of the various heuristic techniques or behaviors necessary to do computer science, but these are some expectations of skills, concepts, and behaviors that professors and students find important in building the experience necessary to become a mature computer scientist. The concepts that professors try to teach their students, especially in lower level classes, often involve learning the syntax of specific programming languages, and how to write programs or functions in those languages. In other classes, concepts of how to build and use different data structures and understand memory management in a specific language are taught with the hope of applying it to what the students do later in their life. This is especially useful as some languages deal with the memory management in computers differently, or problems might occur when changing from one programming language to another. However, they also try to instill in their students more abstract skills and way to solve problems.[85]

[85] Professor Interview, Claremont Colleges

The skills that the professors expect their students to demonstrate include the tangible knowledge of how to code in a different language, but also the understanding of how to approach problems in different ways. Throughout tests, quizzes, homework, and classwork problems, professors attempt to give the students time to not only help build their computational intuition, but also help the students demonstrate what they already have learned. This experience helps develop the computational intuition by building trained judgement within the students. Throughout the homework, the professors expect students to make decisions on how to solve problems, as well as make informed and justifiable choices as they develop their solutions. As the students move through to upper division classes, they are expected to make these decisions while dealing with more complex problems.

The Expectation of Trained Judgement

Throughout the homework assignments, students are expanding their understanding in their current computer science class as well as building off what they have learned from previous classes. Within their assignments, students are often evaluating many different options and ways of approaching the problem, weighing information that they find online, and discussing problems that they have previously solved in hopes of making informed decisions throughout the process of their assignment. Professors expect students to make these choices about various paths they can take through the use of the trained judgment they are developing. As the students move into more advanced classes, they have more experiences and have higher expectations about what they know and their trained judgment.

This ability to maintain an informed decision extends not only to the creation of the students' own answer during practice problems, but includes the ability to analyze other people's code or algorithms. It is especially important as one of the main skills and precepts within computer science is the computer scientists' ability to reuse or recycle code that has already been created, rather than necessarily having to recreate everything from scratch.[86] For example, professors will help students to understand what certain algorithms or pieces of code are better than others for certain situations as well as what other factors may affect coding choices.[87] In many ways, as the computer scientist works at solving the problem, they are using previously solved problems and analyzing which choices will produce the best result for solving a specific problem.

The ability to understand how efficiently a program may run or how to make it run more efficiently is also viewed as an important skill set to develop and is often represented throughout the students' classroom and homework experience.[88] Not only are students expected to know how to examine and analyze an existing algorithm for efficiency, they are expected to analyze the value of their own work. As one professor stated, "there are better answers and less good answers…" and students have to make

[86] Within the education system there are limits to what students can borrow from. The students cannot copy code, but professors recognize internet searches as an important source to understanding how to solve parts of problems, especially with error messages as well as sometimes copying code found on the internet for assignments at the professor's discretion.
[87] A lot of different factors can affect the runtime of a piece of code, these can include how much data you might be looking through, how the data is stored of the computer, what the code is actually doing, and many other factors. When analyzing a piece of code or even when writing your own program, these are all aspects that are necessary to consider during the design phase.
[88] Efficiency within computer science is often about the algorithms (known step-by-step), which relates to the amount of computational resources. The computational resources include aspects such as the number of steps necessary, the amount of storage, memory space, and more. For there to be maximum efficiency of an algorithm, the algorithms normally use the minimum amount of resources necessary.

decisions and judgements about what the best answer is, whether it’s analyzing an existing algorithm or creating their own.[89] In one example the professor had to stress to his class how important it is to be able to choose the better answer:

> “There was a slightly ambiguous thing I gave them, but if you thought about it... it was a task involving binary search trees, and they have all seen that already in either their second year or second class.[90] It is a concept that is familiar and fundamental to computer science. I said implement deletion. I am not going to give you one, look it up, I don’t care. Many of them implemented an algorithm that had the wrong asymptotic, it was too slow, it didn’t do what it was supposed to do. I was so confused, like this obviously wasn’t the right thing… and others were like… well can you have more than one element in the tree. And I was like look at the invariant that I wrote down, look at what I wrote, what do you think, did you consider this? And they were like, oh I guess not…”[91]

In this example, the professor has an expectation of the students to fulfill the task that was given to them. This task was not necessarily well defined, however, the professor had an expectation of the students to have enough trained judgment, that when given a task where they had free reign, they would complete it well enough. The decisions that computer scientists make to code their program, approach their problem, or what other code they may reuse can have drastic effects on how their program runs or how “correct” the professors think the students performed their homework. It is not necessarily sufficient whether or not a specific program might work, as there are choices that students make that can affect the way that the professors and students grade their work.

[89] Professor Interview, Claremont Colleges
[90] Binary search trees are a way of storing data within computer science. After storing the data you are able to lookup, delete, and add more items to the tree.
[91] Professor Interview, Claremont Colleges

Within the classroom environment, professors teach their students methods to analyze code and think about problems so the students are able to create a perception of how good their solutions are on their own. What is taught in class is expected to be utilized during the homework that is assigned. Throughout the students learning, there is hope that the students will be able to work on their homework as a trained expert and to start demonstrating the skills of the "trained expert… [who] grounds his or her knowledge in guided experience…"[92] In the above example from the professor, the students were told to make an informed choice of which algorithm that the student was going to use. Often there might be many different algorithms that a student could use to solve a problem or part of a problem. They might complete the task in a similar way, but some choices are better than others. Through the development of the students' "trained judgment" they are able to make these decisions in more informed and useful ways.

Throughout the computer science education, there is an expectation that the computer scientists will build a computational intuition. As students go on to upper division classes and are required to complete more complex assignments, this knowledge that is built in earlier classes is something they are expected to draw on.[93]

The Judgment

Homework and solving problems in class is one of the ways that both professors and students describe as the main method in which computational intuition is developed. Homework is the exercises where student are able to apply the heuristics they learn or

[92] Daston and Galison, *Objectivity*, 359.
[93] Professor Interview, Claremont colleges

develop their own, as well as the place where students are supposed to demonstrate their trained judgment and ability to make informed decisions throughout the process of working on projects. Throughout the students' education at the Claremont Colleges, these students are building the experience necessary for a successful future within the computer science field, whether this is in industry or academics. Professors try to create situations and problems for students that help them gain experience and develop the necessary problem solving skills. This means providing the students problems that they might actually see if they students go into graduate school or industry.

Although there is knowledge that needs to be taught or transferred to students throughout lectures, the development of computational intuition needs to occur through the experiences that students have through homework and projects. Gaining experience within the field of computer science and knowing when to apply the various different aspects of computer science is important to the creation of the students trained judgment and is central to moving forward as computer scientists. It is this trained judgment, and the heuristics of how to solve problems that students will take with them as their computational intuition as they move forward, and continue to develop and gain more experiences throughout their careers.

Conclusion

Throughout this thesis, I have discussed how the computer scientist is trained and develops their computational intuition. The computational intuition of the computer scientists involves many different aspects, there is what computer scientists call "computer science intuition," computational thinking, heuristics and trained judgment. Computational intuition is taught to the students by their professors as they work together on homework. This way of learning is often social, where the interactions between the professors, tutors, and other students help learners gain an understanding of the "computer science intuition". This characterization of computer science is shaped by how computer science intuition is defined. It helps define what professors teach in classes and eventually how students understand computer science as a field. It is this method of thinking that computer scientists at the Claremont Colleges have stated as being one of the most essential items that should be taught and gained throughout their education.

By understanding computational intuition as an idea of how to solve problems through the use of computers, the computer scientists understanding of the field clearly affects the manner in which computer science is taught and how professors approach it. The focus on homework seems more applicable to understanding this way of thinking, while the classes teach necessary skills and concepts. I will leave you with this final metaphor. Computer science is like a tool box, where the computer scientist has all of their skills and concepts (tools) needed to solve a problem. The computational intuition is how the computer scientist determines which tools they will use from their tool box and how they will use those tools to solve their problem.

The computer science education system is discussed in length, both inside and outside of the Claremont Colleges computer science classrooms. Questions of what makes a successful computer science class and successful computer science students are also important to questions of transferring computational intuition to the next generation of computer scientists. This question of successful computer science classes is one that is being continually asked in today's society and there is increasing interest in computer science education. Classes, though they may teach specific skills and concept within their class periods, are also continuously driving forward this idea of creating a computational intuition in the students

The ability to develop a computational intuition is clearly an important part of the education of a computer scientist. Closer work on understanding the link to the development of computational intuition or the lack of it in those who do not like or drop out of the computer science field over time is needed. Is it the success or lack of development of the "computer science intuition" that contributes to those who leave or stay in computer science for various reasons?

Furthermore, the concept of computational intuition is closely tied to research about how computer science is performed, and how computer scientists solve the problems they are presented with. Though this computation intuition does not necessarily go into the specific subfields within computer science, it may warrant a look at the development of the intuition along those lines. Different subfields might focus on the same ideas of problem solving, building off of the undergraduate understanding, but how are the ways of solving problems changed within these subfields?

This study had several limitations, mainly the short duration over which the fieldwork was completed which in turn could affect the scope of the study itself. One problem arises from the limited number of students within the Claremont Colleges, as the people who I interviewed were almost all computer science majors and did not necessarily capture the view of non-computer science majors taking computer science classes. Within the interviews, students from three out of the five colleges are represented, with students from the other two either not volunteering or volunteering after fieldwork had concluded. Furthermore, fieldwork was limited to introductory classes, at two of the schools. I could not do fieldwork in upper division computer science classes or a class at the third school due to the lack of time and scheduling problems.

The development of the computational intuition is an idea that is closely tied to the overarching goals that many of the computer science departments have when training their students. This computational intuition supports a certain methodology of approaching problems and thinking about them, allowing students to solve problems in computer science. The adoption of the computational intuition by students signals a wider understanding of computer science as a field and the behaviors that are expected of students as they work on their projects, homework, graduate work or enter the workforce.

Bibliography

"Areas in Computer Science", *Cornell University*, accessed August 15. 2015, http://www.cs.cornell.edu/Info/Department/Ugrad/Subareas.html.

Bedner, Anne K., Cunningham, Donald, Duffy, Thomas M., and Perry, David J. "Theory Into Practice: How Do We Link?" In *Constructivism and the Technology of Instruction: A Conversation*, edited by Thomas M. Duffy and David H. Jonassen, Hillsdale, New Jersey: Lawrence Erlbaum Associates, Publishers, 1992. Hillsdale, NJ: Lawrence Erlbaum Associates, 1989. 16-34

Bigger, Maureen, Bauer, Anne, and Yilmaz, Tuba. "Students perceptions of computer science: a retention study comparing graduating seniors with CS leavers", *ACM SIGCSE Bulletin* 40, no 1 (2008): 402-406. Accessed August 15, 2015, http://dl.acm.org/citation.cfm?id=1352274.

Bolton, David. "What is a Compiler?" *about tech*. November 24, 2014. http://cplus.about.com/od/introductiontoprogramming/p/compiler.htm

Brown, J.S., Collins, A. and Duguid, S. "Situated cognition and the culture of learning" *Educational Researcher* 18, no 1 (1989): 32-42. http://www.umsl.edu/~wilmarthp/modla-links-2011/Situated-Cognition.pdf.

Carter, Lorie. "Why students with an apparent aptitude for computer science don't choose to major in computer science." *ACM SIGCSE Bulletin* 38, no 1 (2006): 27-31. Accessed December 10, 2015. *http://dl.acm.org/citation.cfm?id=1121352&CFID=769109340&CFTOKEN=81952656*.

Chaudhry, Nadeem Chaudhry, and Ghulam Rasool. "A Case Study on Improving Problem Solving Skills of Undergraduate Computer Science Students." *World Applied Sciences Journal* 20, no. 1 (2012): 34-39. Accessed March 3, 2016.

Collins, A., Brown, J.S., & Newman, S.E. "Cognitive apprenticeship: Teaching the crafts of reading, writing, and mathematics." In *Knowing, learning, and instruction: Essays in honor of Robert Glaser*, edited by L.B. Resnick, 453-494. Hillsdale, NJ: Lawrence Erlbaum Associates, 1989.

Daston, Lorraine and Galison, Peter, *Objectivity*. New York: Zone Press, 2010.

Dictionary.com, s.v., "Computing", accessed November 1, 2015. http://www.dictionary.com/browse/computer.

Dictionary.com, s.v. "Computer Science", accessed November 1, 2015, http://dictionary.reference.com/browse/computer-science.

Duffy, Thomas M, and Jonassen, David H. Introduction to *Constructivism and the Technology of Instruction: A Conversation*, edited by Thomas M. Duffy and

David H. Jonassen, Hillsdale, New Jersey: Lawrence Erlbaum Associates, Publishers, 1992.

“Exploring Computational Thinking” Google For Education, accessed March 30, 2016 https://www.google.com/edu/resources/programs/exploring-computational-thinking/

Ford, Paul. “What is Code?”, *Bloomberg*, June 11, 2015, http://www.bloomberg.com/graphics/2015-paul-ford-what-is-code/

Gergen, Kenneth J. Relational Being. Cary, GB: Oxford University Press, USA, 2009. Accessed April 3, 2016. ProQuest ebrary.

Greeno, J.G., Collins, A. M., & Resnick, L. B. “Cognition and learning.” In *Handbook of Educational Psychology*, edited by D.C. Berliner and R.C. Calfee, 15-46. New York: Macmillan, 1996.

Kuhn, Thomas, *The Structures of Scientific Revolution*. Chicago: University of Chicago Press, 2012.

Latour, Bruno and Woolgar, Steve, *Laboratory Life: The Construction of Scientific Facts*. Princeton, New Jersey: Princeton University Press, 1986.

Leahy, Paul “Beginner’s Guide to Using an IDE Versus a Text Editor.” *About Tech.* http://java.about.com/od/gettingstarted/a/ideversuseditor.htm

Martinez, Michael E. “What Is Problem Solving?” *Phi Delta Kappa International* 17, no 8 (1998): 605–9. Accessed August 15, 2015 http://www.jstor.org/stable/20439287.

McLeod, S. A. ”Lev Vygotsk.” *Simply Psychology*. www.simplypsychology.org/vygotsky.html

McLeod, S. A. “Jean Piaget.” *Simply Psychology*. www.simplypsychology.org/piaget.html

Merriam Webster Online, s.v. “Intuition” , accessed March 12, 2016. http://www.merriam-webster.com/dictionary/intuition

Merriam-Webster, s.v. “Computer Science”, accessed November 11, 2015, http://www.merriam-webster.com/dictionary/computer%20science.

Polya, George. *How to Solve it: A New Aspect of Mathematical Method.* Princeton: Princeton University Press, 1957.

Plonka, Laura, Sharp, Helen, van der Linden, Janet, and Dittrich, Yvonne. “Knowledge transfer in pair programming: An in-depth analysis.” *International Journal of Human- Computer Studies* 73 (2015), 66-78

Herbert A. Simon, *The Sciences of the Artificial*. Cambridge, Massachusetts: MIT Press, 1981.

Shallit, Jeffery. "A Very Brief History of Computer Science", *University of Waterloo*, 1995. https://cs.uwaterloo.ca/~shallit/Courses/134/history.html.

Styhre, Alexander. "Practice and Intuitive Thinking: The Situated Nature of Practical Work." *International Journal of Organizational Analysis* 19, no. 2 (2011): 109-26. Accessed March 12, 2016. http://www.emeraldinsight.com/doi/abs/10.1108/19348831111135065.

Traweek, Sharon. *Beam Times and Lifetimes: The World of High Energy Physicists*. Cambridge, Massachusetts: Harvard University Press, 1992.

Wimsatt, William. *Re-engineering Philosophy for Limited Beings: Piecewise Approximations to Reality*. Cambridge, Massachusetts: Harvard University Press, 2007.

Wing, Jeannette M. "Computational Thinking," *Communications of the ACM* 49, no. 49 (2006): 33-35. https://www.cs.cmu.edu/~15110-s13/Wing06-ct.pdf.