Jl. of Computers in Mathematics and Science Teaching (2014) **33**(1), 567-592

Beyond Cognitive Increase: Investigating the Influence of Computer Programming on Perception and Application of Mathematical Skills

PETER J. RICH
Brigham Young University, USA
peter_rich@byu.edu

NEIL BLY
Agilix Labs, Inc., USA
neilbly@gmail.com

KEITH R. LEATHAM
Brigham Young University, USA
kleatham@mathed.byu.edu

This study aimed to provide first-hand accounts of the perceived long-term effects of learning computer programming on a learner's approach to mathematics. These phenomenological accounts, garnered from individual interviews of seven different programmers, illustrate four specific areas of interest: (1) programming provides context for many abstract mathematical concepts; (2) programming illustrates the important distinction between understanding the application of mathematics in a specific situation and the execution, in general, of a known procedure; (3) programming habits helped participants divide complex mathematics problems into more manageable tasks; and (4) the need to use mathematics for more efficient programs decreased participants' apprehension and increased their motivation toward mathematics. In short, computer programming provided participants with context, application, structure and motivation for mathematics.

INTRODUCTION

Researchers have long proposed a cognitive connection between learning mathematics and learning computer programming (Clements & Gullo, 1984; Linn & Dalbey, 1985; Palumbo, 1990; Subhi, 1999). A recent report by the National Mathematics Advisory Panel discussing the use of computers in mathematics education encouraged educators to capitalize on that connection:

> Research indicates that learning to write computer programs improves students' performance compared to conventional instruction, with the greatest effects on understanding of concepts and applications, especially geometric concepts, and weaker effects on computation…. The Panel recommends that computer programming be considered as an effective tool, especially for elementary school students, for developing specific mathematics concepts and applications, and mathematical problem solving abilities. (Flawn, 2008, p. 51)

In this paper we argue that although research does indeed support the connection between learning mathematics and learning computer programming (Rich, Leatham, & Wright, 2013), it is primarily the cognitive connections that have been explored. Affective connections are also often claimed, but there is little research to back up these assertions. Herein we report the results of a study in which we gathered phenomenological evidence to examine the *perceived* effect of computer programming on different programmers' approaches to, and understanding of, mathematics.

The Cognitive Influence of Programming on Mathematics

Many cognitive claims for the effect of computer programming on mathematical performance and understanding come from studies conducted using Logo. Papert originally designed Logo to teach children specific mathematics concepts (Feurzeig et al., 1970). In their review of research in which computers were used to influence young children's (birth to grade three) mathematics abilities, Clements, Battista, & Sarama (2001) found programming in Logo increased students' understanding of shape, angle, length, measurement, symmetry, and arithmetic processes, among others. Other studies have shown that the study of Logo has increased students' algebraic reasoning (Clement, Lochhead, & Soloway, 1980), meta-cognitive

abilities (Clements & Nastasi, 1988; Clements & Gullo, 1984), problem-solving ability (Subhi, 1999), and general geometry abilities (Clements, 2002).

The relationship between mathematics and computer programming has also been examined from the other direction—prior background in mathematics has been shown to be a significant variable in the retention of students in introductory computer programming courses (Konvalina, Wileman, & Stephens, 1983). Furthermore, those who perform well in programming courses tend to be those with strong scores in mathematics (Bergin & Reilly, 2006). In these studies, mathematics has continually surfaced as the strongest positive predictor of performance in computer programming (Byrne & Lyons, 2001).

Shortcomings of Computer Programming + Mathematics Studies

Byrne and Lyons (2001) commented that "the link between mathematics ability and programming is widely accepted, although it's empirical demonstration is questionable" (p. 49). The two subjects appear to be treated as having an obvious relationship, almost intuitive in nature. Indeed, Tooke (2001) stated, "there is a trivially obvious connection between mathematics and the computer. It is, in fact, a symbiotic relationship" (p. 2). Such claims lead us to ask *why* mathematics and computer programming are consistently grouped together. In his book *Mindstorms*, Papert (1980) offered this view:

> In my vision, the child programs the computer and, in doing so, both acquires a sense of mastery over a piece of the most modern and powerful technology and establishes an intimate contact with some of the deepest ideas from science, from mathematics, and from the art of intellectual model building. (p. 5)

Claims such as Papert's, that programming creates an "intimate contact" with mathematics, suggest that programming's benefits on mathematics extend beyond increased cognitive abilities. Yet, some of the problems that arise in the literature on computer programming as a way to increase students' mathematics scores suggest that non-cognitive elements have been ignored in the research.

For example, one of the most prevalent omissions in the extant literature is the effect of long-term study of computer programming on mathematical understanding. In many studies, mathematics skills were tested

after a relatively short training period in computer programming, leading to inconclusive or even contradictory evidence among researchers (c.f. Pea & Kurland, 1984). A common criticism was that there was no significant increase in student performance in mathematics after having learned to program. In a review of the effects of programming on learning, Palumbo (1990) offered several insights into possible shortcomings. In some cases, studies failed to account for the time it takes to learn to program. In other cases, researchers failed to test for programming competency in the first place (why would there be transfer when the target skill was never actually learned?). When these concerns were addressed, results were generally positive. For example, Gorman and Bourne (1983) reported that third-grade children programming in Logo for one hour a week performed significantly higher on a test of rule learning than did children with 1/2 hour a week of programming exposure (Clements & Gullo, 1984). De Corte, Verschaffel, and Schrooten (1992) also found that when they accounted for programming competency, they found positive effects on other problem-solving skills.

Palumbo (1990) noted that "longer, more intense exposure should be provided to determine if any generalized transfer will occur" between mathematics and computer programming (p. 81). While no doubt important, the specific focus on establishing a cognitive connection between mathematics and computer programming has ignored the effect learning to program might have on a student's *attitude* toward mathematics and its perceived utility over time. Such research tells us little about what Van Manen (1996) refers to as the *lived experience* of individuals who have learned to program. Understanding how learning to program might motivate a learner to pursue or use mathematics in ways they would formerly not have pursued can be just as compelling an argument to its importance as any cognitive benefit measured through objective assessment of one's resulting mathematical abilities. In this study, we set out to describe the perceived effect learning to program had on varied professionals programmers' understanding of, use and approach toward mathematics over time. In other words, we set out to describe how programmers view the relationship between their own understanding of programming and of mathematics.

METHODS AND MATERIALS

Personal narrative provides a valuable window into understanding the influence programming has had on academic, professional and daily experi-

ences. Research supports that stories are more memorable, and better support learning and understanding than nonstory narratives (Patton, 2002). Concerning storytelling, Mitchell (1979) stated, "our knowledge is made up of the stories that we can tell, stories that must be told in the language that we know…. Where we can tell no story, we can have no knowledge" (p. 13).

Storytelling enables participants to relate detailed instances where they perceive the benefits of programming and the effect this may or may not have had on their mathematical or other understanding. Inasmuch as people's stories reveal the unfolding narrative of their perceived life, we sought to engage participants in telling the story of when they learned to program and other stories of how it affected related events in their lives.

Participants

The research consisted of two phases. An initial snowball survey was posted on several programming forums. Participation in the survey was voluntary and therefore may have revealed a bias toward an interest in the subject. Thus, while we feel the results are honest representations, we do not claim that they apply to all programmers. The survey gathered background information on participants' experience with mathematics and computer programming, as well as their attitudes toward and between the two.

In addition to the reported survey data, we conducted face-to-face interviews to gain a better understanding of the narratives surrounding each programmer's experience in learning to program and how it may or may not have been related to their attitudes toward mathematics. Maximum variation purposeful sampling methods were used to select the research participants in order to gain diverse programming application and experience (Patton, 2002). Only individuals with at least 2-3 years of programming were chosen to participate, though ability in any particular programming language was not a priority. Due to the duration and personal nature of the interviews, availability and local proximity of participants was a practical factor in the candidate selection process. Though there was no intentional gender discrimination and the project allowed for both male and female participants, only 4 individuals out of the 45 that completed the survey identified themselves as females. Of these responses, seven local male participants with diverse cultural, academic, and professional backgrounds were selected for conducting open-ended interviews regarding their experience. Participant cultural and academic diversity spanned the U.S., Canada, England, and

Nepal. Programming backgrounds ranged from self-taught to formal high school and university instruction at both undergraduate and graduate levels. The professional occupations of the selected individuals included Web, mobile, and online education programmers, a geographer, a bioinformatician, and a director of development.

Analysis

As this study focused on individual perception, phenomenological methods were used to analyze the data and interpret the results. Phenomenological analysis seeks to expose the lived experience of an individual or group of people and understand this experience within a particular living context (Crotty, 1998). As Van Manen (1996) stated,

> [The] aim of phenomenology is to transform lived experience into textual expression of its essence—in such a way that the effect of the text is at once a reflexive re-living and reflective appropriation of something meaningful: a notion by which a reader is powerfully animated in his or her own lived experience. (p. 36)

In this approach, it is equally important to highlight the structure of the experience and events as they occurred as well as the individual narrative, or interpretation of events. Thus, the analysis of individual interviews sought to highlight the structure and story of each individual.

Participants were individually interviewed regarding their experiences prior to, during, and following (where applicable) their involvement with programming. These interviews lasted from 1-2 hours and sought to understand the narratives surrounding the influence programming had had on their learning, particularly their learning of mathematics. The interviews were digitally recorded for archival and retrieval purposes and subsequently transcribed for analysis.

Textual and structural analyses were conducted following Moustakas' (1994) recommendations and procedures of pulling out the invariant horizons from a participant's recounting of an experience. Experiences were then reconstructed into narratives that represent the essence of learning to program in relation to its effect (or lack thereof) on mathematics.

RESULTS

We first present the results of the survey, after which we provide our analysis of the 1-on-1 interviews, demonstrating conflicting or confirming views between the two.

Of the 60 respondents who began the survey, 45 completed it fully, with 82.5% reporting that they actively program on a daily basis. The group represented a fairly even spread across many dimensions (see Tables 1, 2 and Figure 1). For example, respondents represented an even spread of ages (from 15 to 40+) as well as level of education (high school GED to PhD). A good portion of respondents (40.9%) were students, while an equal portion self-identified as some type of developer (software engineer, developer, programmer). Most of the remaining respondents were professors or instructors (18.18%), with a single respondent being retired. In addition, though all respondents self-identified as programmers, there was great variability in how many hours they reportedly spent programming each week (see Figure 1). Cumulatively, the group accounted for knowing more than 25 different programming languages.

Table 1
Distribution of Respondents' Current Ages

Answer	# Responses	%
<10 yrs. old	0	0%
14-Oct	0	0%
15-19	7	16%
20-24	12	27%
25-29	8	18%
30-34	7	16%
35-39	3	7%
40+	8	18%
Total	45	100%

Table 2
Highest Degree Earned

Answer	Response	%
GED	1	2%
High School Diploma	11	24%
Associates	3	7%
Bachelors	10	22%
Masters	10	22%
Doctorate	7	16%
None OR Not applicable	3	7%
Total	45	100%

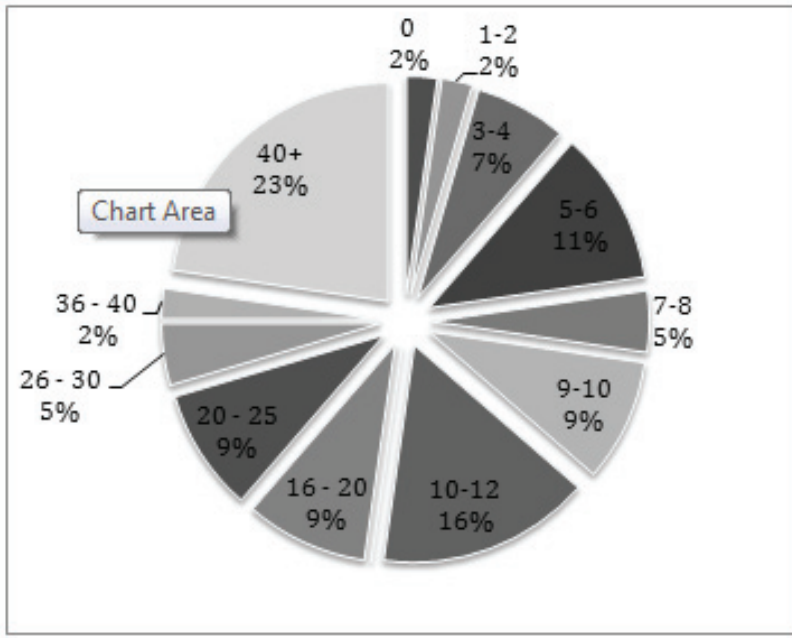

Figure 1. Hours Spent Programming per Week.

Curiously, 30 out of 44 respondents indicated that they had learned to program before 17 years of age. Thus, although respondents themselves covered a wide range of ages, most began programming in their formative years with 10 of those starting at 11 years or younger. When asked how they learned to program, respondents identified several sources, with formal classes, learning from books, and online forums being reported from 68%, 66%, and 41% of respondents, respectively. Overwhelmingly, respondents reported that they learned to program because it was either fun or they found it interesting (53.84%), while another 20.51% began programming to either make or cheat games. Pragmatically, 15.38% were required to learn to program for a class while the remaining 10.26% claimed work opportunity as their reason for learning to program.

In regards to mathematics, respondents largely indicated that they had almost all completed mathematics through the high school level (Algebra 1, 2 and advanced; trigonometry; geometry), with 24 completing calculus during high school. Only 14.29% of respondents indicated that they struggled with or were disinterested in math when younger or before learning to program, while a majority (57.14%) indicated that they had excelled in math in some way. When asked if there were ever a specific instance in which knowing programming helped them with math, 58.82% of respondents responded affirmatively, while 5.88% thought math actually had helped them be better programmers and another 5.88% believing that math and programming were really the same. Curiously, there was no clear trend as to which mathematical concepts most benefitted from learning to program, but instead each affirmative response was accompanied with specific instances of a particular math concept. Some mentioned particular concepts such as angles, functions, summation notation, matrices, proofs and advanced symbolic mathematics; others mentioned particular courses such as trigonometry, geometry and calculus.

INTERVIEWS

Less than half of the survey respondents indicated they would be willing to participate in a face-to-face interview. Of these 22, seven local programmers were interviewed. Each interviewee's final textural-structural narrative was roughly 7-8 pages in length (narrowed from 20-30 pages of transcript each). Due to length restrictions, we are not able to include the full reconstruction here, but introduce each interviewee in a brief description. Although we recognize this is inadequate to fully represent their ex-

perience, their shared insights are more fully discussed, using participants' own words, throughout the discussion. (For full accounts of the interviewee's narratives, see Bly, 2011).

Asim[1]. Originally from Nepal, Asim became interested in programming at a young age. All students in his school were required to learn to program and he "really started liking it" during his very first programming class. By contrast, early on he "had absolutely no care for mathematics". In fact, "math was my second least favorite subject actually…. I tried getting out of homework as much as I could for math, and I was really bad at it." This relationship with mathematics changed when "in the ninth grade I started taking mathematics more seriously. I actually started liking math and then I got really good". When he was in tenth grade, his family moved to the United States, where Asim started working as a programmer. He currently studies bioinformatics and develops games and educational applications.

Ben. Since his youth, Ben was always interested in science. Mathematics did not come easily, however; "It was more of a struggle… and took more time and effort". Ben began programming in college and earned a degree in computer science. He performed "pretty average" in his mathematics-intenstive programming classes, but took comfort that his classmates' mathematical abilities seemd to vary substantially. Although most of his programming experience came from his college courses, "probably 20% or 30% has been through books, not college". He lives in Utah and works as a web programmer. Ben has learned to program in multiple languages including: C, C++, C#, PHP, and Perl. "I program now, and I plan to continue to be a programmer for the rest of my life." Although at this point in his career his work is not math intensive, "in the future, I don't know. It may become really intesnive…. It very well could be".

Brian. Perhaps because he "went skiing just about every day, and did not pay attention in classes," Brian graduated high school with a 2.6 GPA. Brian was not introduced to computer programming until college. "My parents didn't even have a computer until I graduated from high school, so that was a big part of it." He began focusing on academics while in college where he did well in both programming and mathematics. He took several programming classes, in which he excelled, then went on to learn and use Java on his own. He aced three semesters of college Calculus. He enjoyed mathematics and felt it was "more a mindset than anything. It's a quantitative way of looking at things in the universe". He saw both mathematics and programing as essentially problem-solving activities. He earned both bachelors and masters degrees in information systems, and now works as a contract programmer building Internet applications.

1 All names are pseudonyms.

Jason. Originally from Washington State, Jason was banned from attending physical education in middle school because of a fight and other social struggles. He "just sat in the office that period" and all of his academic performance waned, including mathematics. Then some "cool teachers" introduced him to computers and he began bringing programming books to occupy his time while sitting in the office. Once he "got into programming, I realized it kind of re-sparked what I had earlier with math". Jason took his first programming class in 9th grade and it was here that he "got passionate about it and I realized that with programming I could solve problems that hadn't been solved before". He started a web design company when he was 14, seeing "a path for myself to get out of the crumby situation I was in". As an adult he became more interested in social issues, and studied environmental science in college. Jason would like to use technology to help people and improve the world around him. He currently works as a programmer.

Mark. Mark grew up in England, where he preferred "physics, maths, and geography," even though he had to retake several of his mathematics and physics courses. He often asked, "What's the point?" when it came to mathematical calculations, but through applications in the sciences he began to view mathematics as a tool to discover "something you can visualize and apply to a real thing". He came to the United States where he received a master's degree in Geographic Information Systems. While studying, and against the advice of the faculty in his major, he began taking programming classes to enable him to construct digital maps. Although he struggled in those courses he continued to learn programming on his own. Since graduation, he has worked as a computer programmer, and frequently publishes articles concerning geography and technology. Nevertheless, "I never regarded myself as a computer programmer—ever".

Andrew. Andrew grew up in California and Alberta, Canada, and performed well academically. The classes he did the best in were "sciences and probably math class." Andrew had performed well in the mathematics classes he took for his major and considered himself to be a proficient mathematician. Andrew received a bachelor's degree in electronic information technology. While studying electronics, he began taking programming classes. What kept him programming was "the challenge. I always like to think that a person would like programming if they... will spend hours trying to do something only to get a single word to print on a screen, and they think they've accomplished the world". He had not yet faced a mathematical challenge that had impeded his programming, but for the most part he did not believe he "really had to use math other than the basics" in that work. He currently programs web applications that allow other employees to improve efficiency in their work.

Scott. Scott felt two people influenced him along his career path: his father who was an engineer, and his uncle who worked for a university where Scott spent time using computers. His father's influence trained him to solve problems, "so everything that I would encounter, I would look at it and I would try and understand what's happening and then try and fix it, finish it, improve it". He did very well in mathematics classes, even acing BC Calculus while conducting a "sort of social experiment… to show my teacher that I didn't need to do the homework to be successful". Scott's uncle "brought over some programming magazines" and, unlike the homework that he saw as busywork, he spent hours programming the programs from the magazines. Scott was and remains happy to engage in mathematics in the context of programming, but not "when I have to do it over and over and over on a series of problems that don't have anything to do with anything". Scott attended university studying zoology, computer science, and business. At the time of the interview, he worked in the technology department of an online language education company.

DISCUSSION

Four distinct themes emerged across the participants' personal experiences with mathematics and computer programming. Namely, (a) programming provided concrete examples of abstract mathematics, (b) programming provided the "why" to the mathematic "how", (c) programming helped participants learn to break tasks into sub-tasks, and (d) the need to use mathematics for more efficient programs decreased participants' apprehension and increased their motivation toward mathematics. We discuss each of these in the following section.

Contextualization of Abstract Mathematics

Participants shared the viewpoint that programming provided concrete examples of otherwise abstract concepts. Ben recounted, "In the past when I was dealing with logarithms, I didn't quite understand why they were necessary, and then from programming, it's like, that's a good use of them." While trying to calculate the execution time of a function, he discovered the implications of logarithms and consequently "had to go back and learn exactly what a logarithm was . . . When you're doing an algorithm and you want to make it as fast as possible, you're using mathematical concepts to

increase in speed." It was not until Ben had a concrete application of logarithms that the value and practicality of this math concept became apparent. To Ben, programming became "more like a useful tool for math" and concepts were "easier to remember, because I had an actual use for it."

The ability to contextualize mathematics enabled participants to put to use concepts that previously seemed pointless. One survey respondent indicated, "My knowledge of programming has improved my attitude towards math in that it has given me more ways for using it on a daily basis." Illustrating this point, Brian recounted an experience where he used trigonometry to create an accurate clock animation:

> I was creating a clock animation in Flex... but I actually wanted to draw it to the stage [i.e., visually animate the hands of a moving clock], and to do that I had to know the exact mathematical coordinates of where the clock would be placed. And since the clock was refreshing at thirty frames per second, that meant I needed to refresh my angle, and my trajectory, and everything else thirty times per second, three hundred and sixty times a minute, and it [trigonometry] became really important."

In addition to the trigonometry, working in a 3-D environment also required mathematics for "drawing glass buttons, and doing things in 3-D space and trying to make them look spherical." The practicality of this exercise influenced Brian's attitude toward trigonometry: "I think that's what it [i.e., relevant application] does and when people go to school for engineering or something, I think the engineering aspect of things solidifies the mathematical concepts. It makes them actually make sense instead of just being theories." Brian concluded that programming "just helps math be more realistic."

The idea that learning to program gave meaning to mathematics was reiterated by multiple survey respondents. One respondent pointed to a recent example:

> This last week I created a timer class that required a lot of math. It required me to constantly update my measurements based on the angle of a circle. I can truly say that it was one of the first times that doing math inside a circle was enjoyable.

A different respondent shared his experience in learning to program a tank shooter game:

> I realized during the first two weeks of Trigonometry that what we were learning could be applied to a game I had tried (and failed)

> to make over the summer, where a tank would shoot a bullet based on an angle and velocity. I was excited to realize I could use sines and cosines to break the angle and velocity into x and y axes. I never really struggled in math after that point, because I felt it was applicable to my own goals.

Giménez (2004) stated, "The biggest struggle in mathematics is about meaning and significance, which is not revealed by the teaching practice. Why is mathematics to be learned and taught? Why is math applicable?" (p. 145). There are many topics in school mathematics curricula that may elude obvious practical application for students. In early mathematics classes, mathematics often emerges from contexts such as buying apples and oranges at the store, monitoring train arrival times, and calculating distances between cities. As students move into secondary education, however, such practical contexts become further removed from the abstracted mathematics topics at hand and thus more difficult to create. This difficulty often results in problems that "students do not see… as real. The problems often seem contrived and arbitrary and have nonrealistic goals" (Baker & O'Neil, 1994, p. 202). In contrast, programming activities, such as calculating sorting algorithms, provided Asim with what he felt was an appropriate vehicle to better clarify abstract calculus concepts: "Definitely computational time. Calculus really comes in handy when this is converging to this, especially for something like sorting algorithms and everything."

Without programming experience, the participants considered certain areas of mathematics, such as alternate base systems, to be otherwise unapproachable or incoherent. For Scott, "exposure to binary operations [is] a whole field of mathematics that I definitely would have never understood without being involved with programming and computer hardware." Ben supported this sentiment, stating, "Especially with converting from whatever base you're working in, if it's a hexadecimal or a base 2. That is one thing [in programming] that's definitely helped me to understand that [mathematics] concept better."

It is important to note that these programming contexts did not necessarily have a direct relationship to the participants' ease or difficulty in performing the related mathematical operations, but rather it provided a motivational context for the participants to justify learning the concept. Brian said,

> I always thought that geometry and trig were kind of worthless, because I never planned on being an architect, or an engineer, that would actually need to use them. But once I jumped into program-

> ming and had to start using those all the time, I started realizing that it was actually an extremely useful practice to know those things, and I wish that I'd learned them in the first place.

Scott summarized this sentiment when he positioned programming as "connecting the dots" to how mathematics applies to real life. He explained, "Programming has connected the math to real life, whereas before it was fairly abstract. Even when it wasn't abstract, for whatever reason, it just didn't seem as apparent or as necessary."

Why vs. How

Ben described the necessity of going back to relearn the concept of logarithms despite previous study in high school math classes. Without a meaningful application of logarithms at that time, there was no anchor of understanding for Ben to draw on years later. Once the link between concept and application had been created through this programming context, Ben felt that the concept of logarithm "was easier to remember, because I had an actual use for it." The participants in this study seemed to recognize the difference between the ability to perform a particular mathematical procedure and the ability to recognize a situation where that calculation would be needed—the important distinction between execution and application. The National Council of Teachers of Mathematics (1989) stated, "students need to view themselves as capable of using their growing mathematical knowledge to make sense of new problem situations in the world around them" (p. ix.). Participants' experiences with programming seemed to have taught them that, in many cases, simply knowing the "why" is more important than recalling the procedural "how." If one knows what math is needed in a given situation, then the specific process can always be obtained from another source or even calculated by alternative means, as demonstrated by several of the participants who sought out learning materials once they discovered mathematics that was likely to help them solve their programming conundrums.

Scott illustrated this idea with an experience where he and his brother-in-law were working through a problem set and needed to calculate the area of a sphere. He explained to his brother-in-law, "You don't actually have to know and remember the math, or do the math to calculate every sphere, but you have to understand it well enough to write a program to do it for you." Steffe (1990) underscored this effect, explaining that the purpose of "prob-

lem solving is not simply the solution of specific problems. The primary reason is to encourage the interiorization and reorganization of the involved schemes as a result of the activity" (p. 187). Scott continued,

> We're leveraging the power of computers and programming so you don't have to then remember the math. You just know that it's there. You can look at it and figure out how it's working, but you don't ever have to do it.

The significance of this experience is not to say that programming should be thought of as a substitute for learning math, but rather that this experience helped provide an anchor for a particular concept. The programming context created an environment wherein Scott was able to solve a problem using a mathematical concept, but then allow the computer to carry out the mathematical computations.

Asim shared a similar story where he needed to solve a problem in a statistics class:

> I did not want to sit there, I just needed the answer, I just had to fill in the answer, and the error percentage was like five percent or something, so you could be off by five percent, it didn't matter, and, the ideal way to do it was using calculus... and I did not want to sit there and do the calculus.... When I did my calculator stuff in intervals of 1 and just started adding it, but when you go down to like intervals of 0.5, the errors get smaller, 0.05, it gets smaller still. And, this is where the programming skills come in handy, you write a program to do it in intervals of 0.0000001, and then start adding, it gets very, very close to the correct answer.

As a result of the programming, he actually ended up with a much more accurate answer than had he simply carried out the procedure by hand. Curiously, because he had substituted programming in place of a mathematical procedure, Asim felt that he had been deceitful: "That was cheating, kind of. But, yeah, I did get the very correct answer." While it is certainly possible that the class required him to know a particular calculus procedure for solving this problem, in general Asim's approach is just as valid. In everyday scenarios, there is a premium on the ability to know "the" correct procedure to use and why. Furthermore, what he referred to as "cheating the mathematics" is what others may refer to as "mathematical problem solving." Asim experienced "the joy of confronting a novel situation and trying to make sense of it —the joy of banging your head against a mathematical wall, and then discovering that there may be ways of either going around or over that wall" (Olkin & Schoenfeld, 1994, p. 43).

Jason expressed this same sentiment toward using programming to get around a limitation of his mathematical knowledge:

> With programming, I don't need to have a good memory, I have references; I can just quickly access the information that I need to solve the problems, so that is no longer a limitation for me.... I think of programming as an alternative to math. I can't solve this using normal math but I can write a program that will solve it for me.

Jason used programming as an alternative for carrying out mathematical procedures, not a replacement for understanding the underlying mathematical ideas. He felt that programming actually enabled him to make use of mathematics that would otherwise not be at his disposal:

> Programming allows me to use math to help me do what I want to do. I think without it, I'd have less patience for it [mathematics], and I couldn't even see the purpose in it.... I use programming as a tool to help me to use, to kind of harness the power of the math.

Breaking Tasks into Subtasks

Multiple participants independently referred to the game of chess to illustrate how programming helped them consider the effects and possibilities of individual steps in a problem. Brian explained,

> It's like playing chess for your first couple times. You don't think about every possibility that could happen down the road, you just think about your current move. But as you play, you start realizing that if you make this move then this could possibly happen and this could possibly happen and all these things could happen, and your thought to make a single move increases.... And I feel like that's what programming does. It basically just creates a habit of always thinking about what this one choice is going to do, and how it's going to affect every other aspect of the program.

Mark also used a chess analogy:

> It's like playing chess. You try to look moves ahead and think like, "If I do that, and I do that, and I do that, and I do this, what's that going to end up with?" So, I kind of regarded it as a logic exercise and yes, I think being a programmer I think in a more logical way.

The ability to visualize and break problems into smaller sequences was a common response to the open-ended survey question, "in what ways has your knowledge of programming influenced your attitude toward math?" Programming applications generally consists of breaking down complex problems into smaller chunks of subtasks. The programmer then sequences a series of step-by-step instructions to create solutions to these more focused steps. With each step, the programmer is forced to anticipate the consequences of every decision for the individual task and for the problem as a whole. Participants reported that this breaking-down approach influenced their general problem solving habits, as almost any task could be divided into smaller subtasks. Asim stated, "Most importantly, [programming] taught me how to dissect things into smaller chunks."

The participants saw this problem solving heuristic of breaking problems into smaller, logical increments as being extremely valuable in mathematics. Participants reported taking a more systematic approach to mathematics after learning to program. Asim explained that "when you're doing some kind of mathematical algorithm, that [breaking it into increments] is how you approach it." This process of dividing the problem into subtasks has been identified as a valuable skill in mathematics problem solving: "The solver's generation of subgoals and his understanding of the problem situation were intertwined with each other. His settlement of subgoals was supported by his understanding of the situation" (Nunokawa, 2001, p. 203).

The strategy of breaking tasks into meaningful subtasks helped reduce the barriers of more complex problems, allowing participants to create a more manageable path between the problem and the solution. As Catrambone (1998) stated, "A learner possessing an appropriate subgoal, or set of subgoals, will be in a better position to solve new problems compared to learners who memorized only a set of steps" (p. 357). Asim explained,

> But definitely before that [learning to program]... when I approached math I never thought of dissecting the problems into smaller chunks. "Ok, let's derive this first and see where we can go from there." It was never like that. It was always, "How do I get from point A to B?" You really don't think between point A to B there would be A to C, C to D, D to E, E to B.... It's actually usually that way when you're solving math problems..... I think programming taught me how to go to those intermediate points and derive stuff.... I think that helped me."

Brian described a similar connection:

> I think the relationship is that they [math and programming] are both quantitative. With math you're trying to come to a solution

> for a problem, and the route you take to get there is the puzzle or the mystery. That's exactly what you do with programming. You have an end that you're trying to get to in a problem, and a barrier in between, and how you get there is basically the puzzle that helps you get down that path.

In addition, participants felt these problem-solving skills extended beyond mathematics. Ben reported that because of programming, "I definitely take a divide and conquer approach in most everything I do…. I would say [this approach] leaked over into the rest of my life, and I started doing other things that way as well." Scott explained programming's broader influence on his life in this way:

> I'm sure that it [programming] made me more systematic, and possibly more creative, understanding that there's more than one way to solve a problem or to do different things.... I'm sure that it affected me, the way that I thought about it, the way that I attacked it.

Need for Mathematics Decreased Apprehension and Increased Motivation

While programming, participants often needed to actively seek mathematical instruction to solve a programming obstacle. Asim talked of how programming "does force you to learn things [in math]." Scott reported that there were times "where you're trying to solve something programmatically, and you know that it can be solved mathematically. There's often a faster way to solve this problem, if you had better math." In some cases, this phenomenon resulted in the participant approaching their math teachers for direction. Jason explained,

> I'd have programming problems that I didn't have the math yet in my mind to solve, and so I'd go to the next grade math teacher at my school and say, "How do I solve this problem for this program?" I was making one of these brick games, that you bash the bricks with a paddle, like Breakout. I didn't really have the math to understand the angles everywhere, and so I'd go to these teachers and have them help me solve these math problems.

Asim similarly described a situation where he needed assistance while programming a solution for a problem in bioinformatics

> because the algorithm was dependent on math, and I did not have the background for it. I had to go and ask my professor for it, to

> teach me how to do the math behind those things. He taught me and then I could do it.

Imagine two students each being confronted with a new mathematics concept. The first student is exposed to this new concept in a typical mathematics class and, without a practical application to relate to, may find the ideas abstract, difficult or even unnecessary. The student may be learning the math simply to fulfill a school requirement, which does little to impart a sense of importance to the ideas or minimize anxiety and may result in avoidance of situations that require mathematics (Ashcraft, 2002, p. 181). A second student however, finds himself in a situation where he has been trying to solve a programmatic problem and now recognizes the need to learn this same mathematics concept in order to move the project forward. As the necessity for understanding this concept developed organically, there is already a natural sense of relevance surrounding the required mathematics and its application. The student therefore approaches the learning process without trepidation as "what seemed most abstract and distant from the real world turns into concrete instruments familiarly employed to achieve personal goals" (Papert, 1972, p. 353).

For this second student, the necessity of needing to solve an existing problem provided motivation and removed obstructions to learning seemingly difficult mathematics. So often, the solution to students who struggle with a "core" subject such as mathematics is to provide that student with additional time in learning that subject. Such an approach fails to capitalize on convergent cognition, an approach wherein students engage with a complementary subject, which often leads to a firmer understanding of shared underlying concepts and lends greater relevance to the subject (Rich, Leatham & Wright, 2013). Papert (1971) described this phenomenon of using mathematics to achieve specific goals toward a larger purpose:

> To achieve these goals mathematical principles are needed; conversely in this context mathematical principles become sources of power, thereby acquiring meaning for large categories of students who fail to see any point or pleasure in bookish math and who, under prevailing school conditions, simply drop out by labeling themselves "not mathematically minded." (p. 4)

Through their experiences learning mathematics in order to solve programming problems, the participants in this study had also experienced mathematics as a source of power. Brian explained how he is frequently required to learn new math concepts saying,

> Most of the math that I do now is all calculating things within a 3-D space and I never took trig, so anything that has to do with sine or cosine, or any of that type of thing, I've had to figure out on my own.

If this same information had been presented to him in a classroom setting he "probably wouldn't have been interested at all. I feel like a practical application of math always changes my attitude of it."

Several participants echoed this sentiment, that programming improved their attitudes toward mathematics. Ben stated that learning how to program "probably bettered my attitude towards math, in that I had something to use it for." For Ben, programming allowed him to "see things from a new perspective." Jason, in describing where he "would be without programming," explained, "In terms of my math . . . I'd probably hate it more." He added that mathematics is

> really abstract, and seeing the programming, putting it to use, helps me see it.... I think without the programming, I would be lost with the math. I would be more confused about it. Whereas with the programming it's helped me put it to use effectively.

Scott simply stated, "I'm happy to do the math if I'm writing a program that's going to save me work down the line." Asim believed his attitude changed because programming "made math much more interesting." Specifically, learning to program helped Asim view math as simply another set of tools, allowing him to get past intimidating barriers. He explained,

> It [solving math problems] is like programming. In programming you just have this set of tools that you know and you're given this problem that you've never seen before and you solve it.... When you start getting into programming, there are so few rules, and everything else you make up. Same as in math, there are few rules, that you can apply it by however you feel like. Those are the two areas where you can get exposed to that kind of thinking early on: "OK, I have these set of tools, how do I get to this?"

Asim's view of problem solving aligns nicely with Suydam's (1987) observation:

If problem solving is treated as "apply the procedure," then the students try to follow the rules in subsequent problems. If you teach problem solving as an approach, where you must think and can apply anything that works, then students are likely to be less rigid. (p. 104)

Asim felt the freedom to use tools that "helped me to do math. It makes you kind of less scared of math. People are scared of math."

IMPLICATIONS AND CONCLUSION

This study found that programming increased participants' utilitarian views of mathematics, and by extension, their attitudes. Research in mathematics education discusses similar issues with "the bridging process between everyday practices and school mathematics" (Presmeg, 2002, p. 295). Dossey (1992) suggested that "a philosophy [of mathematics] should call for experiences that help mathematician, teacher, and student to experience the invention of mathematics. It should call for experiences that allow for the mathematization, or modeling, of ideas and events" (p. 42). Programming provided a rich environment where difficult mathematics concepts and easily forgotten mathematical procedures could be explored, reinforced, and appreciated. Studies such as Moses and Cobb (2001) advocate project-based approaches to mathematics instruction, "using a version of experiential learning" where "each step is designed to help students bridge the transition from real-life to mathematical language and operations" (p. 119). Considering the positive experiences of participants in this study, further study on utilizing programming projects to facilitate these experiences and help overcome contextual obstacles should be investigated.

The participants in this study reported improved attitudes, increased motivation, and decreased anxiety toward mathematics as beneficial effects of programming. Wilkins and Ma (2003) stated, "A person's mathematical disposition related to her or his beliefs about and attitude toward mathematics may be as important as content knowledge for making informed decisions in terms of willingness to use this knowledge in everyday life" (p. 52). Although not typically considered standard instruction, future studies might explore adding computer programming to school curriculum, potentially leading to improved mathematics understanding, self-efficacy, and decreased anxiety. In contrast to the typical American experience, growing up in Nepal, Asim felt his early exposure to programming in school increased his attitude and approach to problems in concurrent math classes. In a similar vein, Hembree (1990) warned of ignoring the consequences of mathematics anxiety:

> Despite its lack of independent identity, the research of mathematics anxiety has prospered, spurred by increasing perceptions that the construct threatens both achievement and participation in mathematics. These suggestions have national import; when otherwise

> capable students avoid the study of mathematics, their options regarding careers are reduced, eroding the country's resource base in science and technology. (p. 34)

Although the participants in this study were able to provide many examples of applications and uses of mathematics, none would describe themselves as mathematicians. In Mark's extreme case, although he identified himself as a programmer and problem-solver, he explicitly distanced himself from mathematics, even after using advanced mathematical concepts to solve programming problems. This finding suggests that a possible misunderstanding of what it means to be a mathematician is being perpetuated either academically or culturally. Dossey (1992) explained:

> The conception of mathematics held by the teacher may have a great deal to do with the way in which mathematics is characterized in classroom teaching. The subtle messages communicated to children about mathematics and its nature may, in turn, affect the way they grow to view mathematics and its role in their world. (p. 42)

There may be failure among students and professionals to recognize certain types of problem solving as mathematics, perhaps mistaking the procedural aspects of mathematics for the whole, ignoring the deeper conceptual ideas and applications. Studies suggest children typically hold a fundamentally narrow viewpoint that mathematics is primarily computation and arithmetic (Fuson, Kalchman, & Bransford 2005; Grootenboer, 2003; Young-Loveridge, Taylor, Sharma & Hawera, 2006). Further research should be done to investigate ways that these views of mathematics are perpetuated academically, culturally, and practically. As Young-Loveridge (2006) stated:

> Given that mathematics is part of the core curriculum, we think it could be important for teachers to help children engage with ideas about the nature of mathematics. In preparation for such discussions, teachers might benefit from examining their own beliefs about what mathematics is, and reflecting on the subtle messages they might convey to students about the nature of mathematics. (p. 589)

This study revealed the potential effects learning to program could have on one's perception and application of mathematics. The results of the research pointed to four specific areas of interest. First, learning to program provided context for many abstract concepts. Second, programming illustrated the important distinction between understanding application of math

in a specific situation and the execution of a known procedure. Third, programming habits helped participants divide complex problems into more manageable tasks. Finally, the necessity of solving a programming problem provided motivation and eliminated apprehension toward mathematics. The message about mathematics from these programmers is an important one to hear—programming provided a reason to learn mathematics. Programming gave these participants context, application, structure and motivation in their study of mathematics. Would that all students of mathematics had such means.

References

Ashcraft, M. H. (2002). Math anxiety: Personal, educational, and cognitive consequences. *Current Directions in Psychological Science, 11*(5), 181-185. doi:10.1111/1467-8721.00196

Baker, E. L., & O'Neil, H. F. (1994). *Technology assessment in education and training.* Hillsdale, NJ: Psychology Press.

Bergin, S., & Reilly, R. (2006). Predicting introductory programming performance: A multi-institutional multivariate study. *Computer Science Education, 16*, 303-323.

Bly, N. M. (2011). Investigating the Influence of Computer Programming on Perception and Application

of Mathematical Skills. Unpublished Master's thesis. Retrieved 18 Nov. 2013 from http://contentdm.lib.byu.edu/cdm/singleitem/collection/ETD/id/2691/rec/1.

Byrne, P., & Lyons, G. (2001). The effect of student attributes on success in programming. *SIGCSE Bulletin*, 33(3), 49-52. doi:10.1145/507758.377467

Catrambone, R. (1998). The subgoal learning model: Creating better examples so that students can solve novel problems. *Journal of Experimental Psychology, General, 127*, 355–376.

Clement, J., Lochhead, J., and Soloway, E. (1980). Translating between symbol systems: Isolating common difficulties in solving algebra word problems. *COINS Technical report*, p. 79-119, Department of Computer and Information Science, University of Massachusetts, Amherst.

Clements, D. H. (2002). Computers in early childhood mathematics. *Contemporary Issues in Early Childhood*, *3*(2), 160-181.

Clements, D. H., Battista, M. T., & Sarama, J. (2001). Logo and Geometry. *Journal for Research in Mathematics Education, Monograph 10*. Reston, VA: National Council of Teachers of mathematics. doi:10.2307/749924

Clements, D. H., & Gullo, D. F. (1984). Effects of computer programming on young children's cognition. *Journal of Educational Psychology, 76*, 1051-1058.

Clements, D. H., & Nastasi, B. K. (1988). Social and cognitive interactions in educational computer environments. *American Educational Research Journal, 25*(1), 87–106.

Crotty, M. (1998). The foundations of social research: Meaning and perspective in the research process. London, England: Sage Publications.

de Corte, E., Verschaffel, L., & Schrooten, H. (1992). Cognitive effects of learning to program in logo: A one-year study with sixth graders. In E. de Corte, M. C. Linn, H. Mandl, & L. Verschaffel (Eds.), *NATO ASI Series. Series F: Computer and Systems Sciences: Vol. 84. Computer-Based learning environments and problem solving.* (pp. 207-28). New York, NY: Springer-Verlag.

Dossey, J. (1992). The nature of mathematics: Its role and its influence. In D. A. Grouws (Ed.), *Handbook of research on mathematics teaching and learning* (pp. 39-48). New York, NY: Macmillan.

Feurzeig, W., Papert, S., Bloom, M., Grant, R., & Solomon, C. (1970). Programming-languages as a conceptual framework for teaching mathematics. *SIGCUE Outlook*, 4(2), 13-17. doi:10.1145/965754.965757

Flawn, T. (2008). Foundations for success: The final report of the National Mathematics Advisory Panel. Washington, DC: U.S. Department of Education.

Fuson, K. C., Kalchman, M. & Bransford, J. D. (2005). Mathematics understanding: An introduction. In M. S. Donovan & J. D. Bransford (Eds.), *How Students Learn Mathematics in the Classroom* (pp. 217-256). Washington, DC: National Academies Press.

Gorman, H., & Bourne, L. E. (1983). Learning to think by learning logo: Rule learning in third-grade computer programmers. *Bulletin of the Psychonomic Society, 21*(3), 165-67.

Grootenboer, P. J. (2003). The affective views of primary school children. In N. A. Pateman, B.

J. Doughtery, & J. Zillioz (Eds.), *Navigating between Theory and Practice* (Proceedings of the 27th conference of the International Group for the Psychology of Mathematics Education, Vol. 3, pp. 1-8). Honolulu: University of Hawai'i.

Hembree, R. (1990). The nature, effects, and relief of mathematics anxiety. *Journal for* Research In Mathematics Education, 21, 33-46.

Konvalina, J., Wileman, S. A., & Stephens, L. J. (1983). *Math proficiency: A key to success for computer science students.* Commun. ACM, 26(5), 377-382. doi:10.1145/69586.358140

Linn, M. C., & Dalbey, J. (1985). The cognitive consequences of programming instruction in classrooms. *Educational Researcher, 14*(5), 14-29.

Mitchell, R. (1979). *Less than words can say: The underground grammarian* (1st ed.). Boston, MA: Little, Brown.

Moses, B., & Cobb, C. (2001). *Radical equations: Math literacy and civil rights.* Boston, MA: Beacon.

Moustakas, C. (1994). *Phenomenological research methods* (1st ed.). Thousand Oaks, CA: Sage Publications.

National Council of Teachers of Mathematics (1989). *Curriculum and evaluation standards for school mathematics*. Reston, VA: Author.

Nunokawa, K. (2001). Interactions between subgoals and understanding of problem situations in mathematical problem solving. *Journal of Mathematical Behavior, 20*, 187-205. doi:10.1016/S0732-3123(01)00069-4

Olkin, I. & Schoenfeld, A. (1994). A discussion of Bruce Reznick's chapter. In A. Schoenfeld (Ed.), *Mathematical thinking and problem solving* (pp. 39-51). Hillsdale, NJ: Lawrence Erlbaum Associates.

Palumbo, D. B. (1990). Programming language/problem-solving research: A review of relevant issues. *Review of Educational Research, 60*(1), 65-89.

Papert. S. (1971). *Teaching children to be mathematicians vs. teaching about mathematics*. Cambridge, MA: MIT Artificial Intelligence Laboratory.

Papert, S. (1972). Teaching children thinking. *Innovations in Education & Training International, 9*, 245-255. doi:10.1080/1355800720090503

Papert, S. (1980). Mindstorms: Children, computers, and powerful ideas. New York, NY: Basic Book.

Patton, M. Q. (2002). *Qualitative research & evaluation methods* (3rd ed.). Thousand Oaks, CA: Sage Publications.

Pea, R. D., & Kurland, D. M. (1987). On the cognitive effects of learning computer programming. *Mirrors of minds: patterns of experience in educational computing* (pp. 147-177). Ablex Publishing. Retrieved from http://portal.acm.org/citation.cfm?id=30914

Presmeg, N. (2002). Beliefs about the nature of mathematics in the bridging of everyday and school mathematical practices. In G. Leder, E. Pehkonen, & G. Torner (Eds.), *Beliefs: A hidden variable in mathematics education?* (pp. 293-312). Dordrecht, Netherlands: Kluwer.

Rich, P., Leatham, K., & Wright, G. (2013). Convergent cognition. *Instructional Science*, 41, 431-453. Doi: 10.1007/s11251-012-9240-7

Steffe, L. P., & Wood, T. (Eds.). (1990). *Transforming children's mathematical education.* Hillsdale, NJ: Lawrence Erlbaum.

Subhi, T. (1999). The impact of LOGO on gifted children's achievement and creativity. *Journal of Computer Assisted Learning, 15*(2), 98-108. doi:10.1046/j.1365-2729.1999.152082.x

Suydam, M. (1987). Indications from research on problem solving. In F. R. Curcio (Ed.), *Teaching and learning: A problem solving focus*. Reston, VA: National Council of Teachers of Mathematics.

Tooke, D. (2001). Introduction. *Computers in the schools*, *17*(1), 1-7. doi:10.1300/J025v17n01_01

van Manen, M. (1996). *Researching lived experience: Human science for an action sensitive pedagogy*. Albany: State University of New York Press.

Wilkins, J. L. M., & Ma, X. (2003). Modeling change in student attitude toward and beliefs about mathematics. *The Journal of Educational Research, 97*(1), 52-63.

Young-Loveridge, J., Taylor, M., Sharma, S., & Hawera, N. (2006). Students' perspectives on the nature of mathematics. In P. Grootenboer, R. Zevenbergen, & M. Chinnappan (Eds.), *Identities, cultures and learning spaces: Proceedings of the 29th annual conference of Mathematics Education Research Group of Australasia* (Vol. 2, pp. 583- 590). Sydney, Australia: MERGA.