SARAH CARRUTHERS, TODD M. MILFORD, YVONNE COADY, CELINA GIBBS, KATHERINE GUNION AND ULRIKE STEGE

6. TEACHING PROBLEM SOLVING AND COMPUTER SCIENCE IN THE SCHOOLS

Concepts and Assessment

Computer science is no more about computers than astronomy is about telescopes. (Dijkstra, n.d.)

Computer Science Education (CSEd) is a young field that is comprised of numerous established disciplines, such as science, mathematics, education, and psychology. Fincher and Petre (2004) in their seminal text on CSEd suggested that moving the discipline toward independence would require that researchers ask questions that may only be answered through computer science. Because of CSEd's relative youth, it is common for researchers in this problem space to look to other disciplines for theory to help answer research questions. This chapter outlines pilot studies that exposed middle school students (11 to 13-year-olds) to a series of new and unique hands-on curricula associated with numerous fundamental concepts in Computer Science (CS). We hypothesized that through experiences for youth with activities such as those outlined here the number of students who understand the concepts covered and who might potentially pursue CS in postsecondary education will increase. In this chapter, the curriculum, classroom experiences, preliminary (largely descriptive and qualitative) results, and next steps in our research are discussed.

The importance of attracting students to the discipline of CS is well recognized (Carter, 2006; Flakener, Sooriamurthi, & Michalewicz, 2010; Klein, 2006; Slonim, Scully, & McAllister, 2008; Zweben, 2008). There is a severe shortage of individuals pursuing CS at the postsecondary level—the essential pipeline issue. Since the CS industry's peak around 2000–2001, CS departments in universities and colleges across Canada have experienced declining enrolment (Slonim et al., 2008). At the same time, CS graduates have emerged into a job market with a higher projected demand level than the average in other areas (1.9% vs. 0.7%) and a higher percentage making above \$50,000 annually compared to other occupations (52% vs. 31.1%; Service Canada, 2011).

While it appears that this trend of declining enrolment may be reversing in the USA, the same is not yet certain in Canada (Zweben, 2008). However, there continues to be concern with enrolment in CS at the postsecondary level. Thus, attracting more students to the CS discipline will potentially improve enrolment and retention in postsecondary CS programs and increase the number of graduates and will more closely match industry's needs. It is hypothesized that this enrolment concern is the result of a perception of a decreased number of jobs in the industry, inaccurate perceptions of what computer scientists do, and general unfamiliarity with the content of the discipline (Carter, 2006; Klein, 2006). However, Flakener et al. (2010) take this idea further to suggest that this is even more serious as "today's marketplace needs more skilled graduates capable of solving real problems of innovation in a changing environment" (p. 20). What they believe is missing from the majority of CS curricula is a focus on developing problem-solving skills. To address this concern, we have focused our research interests on the teaching of problem-solving skills.

There are many approaches to addressing the pipeline issue and to improving enrolment and retention in CS programs, including informing students about CS and better preparing students before they enter undergraduate programs (Slonim et al., 2008). This solution would address both the pipeline issue and the mainline issue of seeking to improve citizens' general CS understanding (Yore, Chapter 2 this book). Various strategies for improving elementary, middle, and secondary school CS education are suggested, including supporting CS teacher education and improving curriculum by better defining what constitutes the CS knowledge base (Goode, 2008). While these efforts are laudable, elementary mathematics and science curricula should also be examined to identify where elements of CS can be infused to support these subjects before secondary school, a seemingly appropriate strategy for this highly interdisciplinary area. As in all fields, we should constantly reflect upon and re-evaluate what we are teaching. By strengthening the foundation of CSEd, we will have a more educated student body around the fundamental CS concepts as well as one that better understands how CS fits into society (i.e., an understanding of what computer scientists do).

ELEMENTARY SCHOOL COMPUTER SCIENCE

The question is whether or not CS should be taught in elementary schools. The current prescribed elementary school curriculum is overcrowded and continues to expand with requests to add new disciplines (e.g., environmental education, engineering design, etc.; see Yore, Chapter 2 this book) while instructional time in the school year is not increasing. At the 2010 Western Canadian Conference on Computer Science Education in Kelowna, British Columbia (BC), critics of the introduction of CSEd in elementary classrooms pointed out that, while students today ride in automobiles, they are not taught automotive mechanics in elementary classrooms and, therefore, being surrounded by computational devices, computers, and other information communication technologies does not imply that they need to be taught CS. However, while the elementary science curriculum in Canada does not contain automotive mechanics, it does include units in physics and chemistry—the fundamental sciences that allow us to understand how and why automobiles work. Similarly, elementary students ought to be able to learn some foundational elements of how and why computers work as well as the use of CS to facilitate problem solving. Carruthers (2010a) noted that most youth currently use technological devices on a daily, if not hourly, basis; communication via cell phones or instant messenger programs on laptops and desktops are examples. Internet access via mobile devices is on the rise, and mobile phone use is overtaking traditional phone networks worldwide. These changes in technology use happen rapidly and have implications for personal privacy protection (e.g., Facebook$^{©}$). Basic CSEd, integrated into today's curriculum, can be the basis for understanding how these technologies work, how this may be possible (Carruthers, 2010b; Romanow, Stege, Agah St. Pierre, & Ross, 2008).

Perhaps even more important, CSEd has broad benefits. Beyond being simply an intellectual pursuit, CSEd teaches problem solving, supports and connects to other sciences and disciplines, can be engaging for different types of learners, and can lead to many career paths other than software developers or designers. *A Model Curriculum for K–12 Computer Science* (United States Computer Science Teachers Association [CSTA], 2006) stated that, "Professionals in every discipline—from art and entertainment, to communications and health care, to factory workers, small business owners, and retail store staff—need to understand computing to be globally competitive in their fields." (p. 11). Giving young students an understanding of basic CS and nurturing an interest in the field is beneficial not only for students who go on to pursue a major or career in CS (pipeline goal) but also for all students (mainline citizenship goal). Further, "Basic computer science education can provide world citizens with the tools and knowledge necessary to make informed decisions about how to use technology, and how to share information" (Carruthers, 2010a, p. 7).

THREE CORE CONCEPTS IN COMPUTER SCIENCE

For each of the three studies detailed below, we selected a CS core concept: concurrency, recursion, and graph theory. These concepts are in line with those identified as required in the model K–12 CS curriculum (CSTA, 2006). Each concept plays an important role in basic problem solving and, once learned, enriches students' general problem-solving skill set. Further, all play an important role in algorithm development and programming. These studies are presented in the order in which they were conceptualized and conducted. The collaboration between Computer Science and Education was established midway through both the concurrency and recursion studies and was only fully involved in all aspects of design, implementation, and analysis of the graph theory study. This increased collaboration will be reflected as the reader moves through the studies; we are excited about the future possibilities this collaboration holds.

Our approach to presenting the CS concepts in the three studies is decidedly constructivist (Yore & Van der Flier-Keller, Chapter 1 this book). We accepted the idea of constructivism in our program design that views individuals as actively involved in constructing knowledge for themselves (Schunk, 2000). The core concept of constructivism is that knowledge is built by learners rather than simply transmitted among persons. In Education, there is increasing understanding and acceptance of the value in students working together to construct meaning about subject matter, resulting in student groups constructing more complex understanding of a topic than any single student could do alone (Ormond, Saklofske, Schwean, Harrison, & Andrews, 2006). In all activities, the students were provided with materials with which they could become actively engaged through manipulation and social interaction. Some of the ways we sought to aid students to construct their knowledge base was through experimenting with challenging activities, negotiating in small groups, encouraging the presentation of individual and group ideas to others, and emphasizing conceptual understanding by focusing on a few core topics then exploring them in detail. We did this by using authentic activities and resources, promoting dialogue with peers in the sessions, and creating a community of learners to aid in learning.

METHOD

We used a lesson study design to explore and improve the teaching and instructional resources (Bell, Witten, Fellows, Adams, & McKenzie, 2006). The lessons focused on an important CS concept related to problem solving that did not necessarily require computer technologies. The preliminary lessons were taught to a target group or intact class of elementary school students. Observations and field notes of the instruction, students' actions, and outcomes were used to document the initial teaching and learning. These data were interpreted to make inferences about lesson effectiveness and improvements needed regarding the students' actions, understanding, and use of the CS concepts. Lessons were modified to address the strengths and weaknesses identified in the first cycle of lesson study. However, in this chapter, we report only the preliminary lesson design and study results.

Lesson Study 1: Concurrency

Concurrency may be best described as acting together either as events, circumstances, or agents. In CS, concurrency describes a property of systems: several computations executed simultaneously while potentially interacting with each other. These computations may happen on the same processor or on separate processors. Concurrency has achieved more importance with the arrival of a new era of programming where developers must consider the subtleties of concurrency inherent in modern many-core architectures. This calls for a revamp of the area, ranging from fundamental pedagogical processes to software development tools. An observed problem with teaching, learning, and using concurrency is that corresponding real-world scenarios, commonly leveraged in pedagogical practices, contain implicit relationships that are difficult to explicitly anticipate in complex code-bases. In our study on the concept of concurrency, we observed students' abilities to come up with both valid and creative solutions to the assigned problems addressing these issues. We highlight three activities and various students' solution approaches to investigate ways to introduce key concepts to Grade 7 students (Gunion, 2009).

All three activities used real-world scenarios or a storyline style approach. The first activity, the dishwashing scenario, is about two people washing a set of dishes: a stack of dirty dishes needs to be cleaned, dried, and put away. The second activity, the movie ticket scenario, involves two people and two ticket queues: two friends wishing to purchase nonrefundable tickets for a popular movie at a theatre with two long queues. The movie theatre has two cashiers—each positioned in such a way that people in the two different queues cannot see each other. The movie has begun and the students need to decide if they will split up to try to buy tickets. The third activity, the dining philosophers' scenario, is an altered version of the classic pedagogical, concurrency scenario of the dining philosophers: here, five knights are sitting at a round table with a single fork between each two neighbours. Each knight requires the acquisition of two forks in order to eat. Students were asked to come up with solutions to manage the forks shared between the knights.

Each scenario allows a range of concurrency to be introduced within the possible solutions for completing the activity. While different strategies for solving the problem(s) introduce different levels of complexity in executing a solution, the core activity remained the same. We refer to the general notion of tasks associated with each activity as *computation*. With the introduction of multiple students participating in each activity, some level of *communication* between individuals was required while solving the activities' problem. The following subsections indicate the ways in which computation tasks and communication play out within the solution space defined by the students for each scenario. We studied the two core concepts of communication and computation individually as well as their interaction, *concept overlap*. Further, we discuss some of the associated consequences encountered when concurrency is introduced.

Scenario 1: Dishwashing. Although the students proposed a variety of solutions for distributing the work, the core task remained the same with differing communication levels in each case.

Computation The subtasks of washing a dish, drying a dish, and putting a dish away—for all of the dishes in the stack—were immediately identified by all students as core elements of computation. All of these subtasks can be considered small, distinct pieces of computation that combine to make up the larger complete task.

Communication The exact communication between participants in this scenario is solution-dependent. In general, students identified a visual communication mechanism that would occur between participants as one person handing off a dish to the next person. While the concrete hand-off point varied, the idea that one person must complete a portion of the task before the next person can receive that dish prescribed a partial ordering of the subtasks.

Concept overlap Although students seemed to immediately recognize elements of computation and distinguish them from elements of the communication process, both concepts are tightly coupled; namely, the act of completing one computation is a form of communication to a partner.

Scenario 2: Movie Tickets. This scenario is a slightly concocted problem in which communication is limited by lack of visual contact between the queues. Students had to come up with solutions that involved sticking together or splitting up while considering the consequences of each.

Computation The computation in this scenario is simply the act of purchasing a ticket. Depending on the solution, multiple tickets can be bought by one person or a single ticket by each person.

Communication Communication was necessary in cases where students decided the quickest way to get tickets was to split up. They soon realized the associated consequence of possibly buying too many tickets. This required them to consider creative ways to communicate beyond subtle visual cues.

Concept overlap The overlap between computation and communication in this scenario is solution-dependent. In the case of students using two ticket queues, computation can be performed at the two ticket booths but this introduces the need for communication. In the case of the students staying together in one queue, communication is not necessary; however, the concurrency is sacrificed.

Scenario 3: Dining Philosophers. The students discussed whether a time exists when every knight can eat to avoid starvation. At any one time, a knight either eats—requiring access to the forks on their left and right at the same time—or thinks.

Computation The tasks are the actions of eating and thinking—although attempting to acquire a fork is arguably a subtask as well. However, this task is considered a second priority task.

Communication Communication is aided by the visual cue of a fork being available for use by a knight wishing to transition from thinking to eating. This communication of one participant attempting to acquire a fork from two adjacent participants is required to allow a knight to eat.

Concept overlap Similar to the dishwashing scenario, visual cues (e.g., an adjacent knight releasing or acquiring a resource) prescribed an ordering such that the end of one subtask could trigger the beginning of another although this was not a precondition for the subtask. Students encountered classic CS concurrency issues (e.g., race conditions, deadlock, starvation, etc.) through role playing and devised multiple strategies for managing the shared resource to facilitate each participant's ability to eat. Each strategy required some form of communication between participants based on the computation (e.g., eating, thinking) patterns.

The three pedagogical activities in Lesson Study 1 helped identify common ways in which these students thought about problems associated with concurrency. In each case, students immediately identified tasks or subtasks and coupled them with the often implicit communication mechanisms, including visual cues.

Lesson Study 2: Recursion

Recursion (Goodrich & Tamassia, 2002; Grimaldi, 1998) is a mathematical concept as well as a CS programming construct that can serve as a problem-solving strategy.

It is a method commonly used to approach and solve numerous problems (e.g., searching and sorting tasks) and consists of defining functions where the function being defined is called within its own definition until a stop condition (i.e., base case) is met. Recursion is observed in the built and natural worlds (Briggs, 1992); for example, virtual advertisements such as the Borax Soap Box (http://piscines-apollo.com/images/borax_box.jpg), a picture within a picture often referred to as the Droste effect. This common occurrence of recursion within day-to-day experiences of many people, including the middle school students in this study, adds to the appeal, relevance, and application of the concept.

Because of its ubiquity and usefulness, recursion is usually taught in its most basic form in the first 2 years of programming (Hsin, 2008). When and how to teach recursion has long been a topic of debate within CS because it is both central in the discipline and thought to be difficult to learn and comprehend (Haberman & Averbuch, 2002). Once recursion is understood, it may be applied to many traditional CS problems. There is a rich literature base documenting efforts to improve the teaching of recursion at the elementary CS level. Some CSEd researchers have suggested that (a) expanding the instructional approach toward a nontraditional, more active and inclusive one would be worthwhile and (b) providing outreach activities beyond university classes would have the added benefit of appealing to nontraditional and underrepresented populations in CS (Ford, 1982; Goode, 2008). This approach to instruction guided our lesson study.

We conducted a pilot study with students in an after-school program (SPARCS, http://outreach.cs.uvic.ca/) that sought information on an appropriate and successful set of engaging hands-on lessons in recursion for a small (n = 9) group of Grades 6–8 students aged 11 to 13 years (Gunion, 2009; Gunion, Milford, & Stege, 2009a, 2009b). We focused on three basic types of recursion: head recursion, tail recursion, and divide and conquer. The activities designed to teach recursion investigated a combination of kinaesthetic learning and programming activities. These activities were similar to those used by Bell et al. (2006) and involved hands-on, problem-solving, group activities focused on CS basics without the direct use of technology, called *unplugged*. A series of weekly unplugged activities was developed to convey the mental models needed, building upon topics that were covered in the previous weeks. Each lesson attempted to explore at least one of the main questions: Can students (a) recognize recursion, (b) understand recursion, and (c) develop a more positive attitude toward CS and recursion? Furthermore, students were asked to solve programming activities in the LOGO-based language (MicroWorlds EX, 2010) to deepen and apply their understanding of recursion in a programming context.

Using a variety of data collection tools (i.e., clickers, written answers, video and audio recordings), the pilot study results found middle school students increasing their recognition of recursion and enjoyment of these activities and, in some cases, their ability to problem solve with recursion improved. We were able to show on this small scale that exposing these students, long before they find themselves in first year university CS, may ease the challenge that both students and instructors face when covering this topic. Furthermore, after being introduced to similar concepts in ways that foster enjoyment and interest, we may find more and better suited undergraduates studying CS at university—an outcome that would benefit the students, the university, and society.

Lesson Study 3: Graph Theory

One important area based in theoretical CS is graph theory. Graphs—not to be confused with the more commonly known bar or line graphs—are a mathematical construct that can be used to model relationships and connections. A graph is a set of vertices and a set of edges. Vertices (typically indicated by a dot or circle) can be connected by edges (represented by lines) to indicate a relationship or connection between them. An evolutionary tree is an example of a graph, where species are represented by vertices and evolutionary connections are represented by edges. Another example is a road map, where roads connect cities and towns (see Carruthers, 2010c, for more examples).

Graph theory has its origins in the historical Seven Bridges of Königsberg Problem proposed by Leonard Euler in 1735. Parts of the city of Königsberg (including two islands) are divided by a river and connected by seven bridges (Figure 1 left). The question posed is: Is there a way to walk through the city and cross every bridge only once? This problem can be modeled as a graph with the distinct land masses (the two islands and the mainland on either side of the river) represented by vertices and the bridges as edges (Figure 1 right).

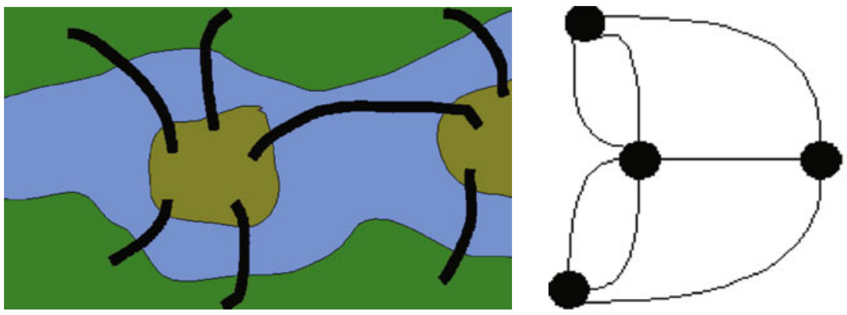

Figure 1. Euler's problem of the seven bridges of Königsberg and a graph model of same (Carruthers, 2010a).

Graph theory investigates properties of graphs such as paths, cycles, topology, and connectivity. Graphs are used to represent data structures in computer programs. For example, file systems that navigate and search for files on a computer are often modeled as a graph, where files and folders are represented by vertices and edges indicate the hierarchy or an *is in* relationship—a file is in a folder that in turn is in another folder. Findings in theoretical CS lead to better, more efficient methods for accessing and manipulating files and folders in these file systems. This connection between data structures and graph theory is important and is an example of one of the many ways in which a sound understanding of theoretical CS can lead to better programmers and more user-friendly computer programs. Like many other areas of theoretical CS, graph theory has connections to other disciplines including biology (modelling habitat and migration patterns) and sociology (analysis of social networks). Using graph theory in problem solving is particularly important. Relational graphs provide a means to distil a problem, highlighting only the relevant information. Extracting relevant data from a problem is fundamental to successful problem solving.

In a recent pilot study investigating the impact of graph theory instruction in Grade 6 mathematics classes, students were taught the basics of graph theory, including how to correctly construct a graph, and how to abstract a problem to a graph representation (Carruthers, 2010a; Carruthers, Milford, Pelton, & Stege, 2010). Participants learned the basics of graph theory and, on certain types of problems, to adopt graphs as a problem-solving strategy. Additionally, for some students, there was a positive association between the use of graphs in solving problems and the correctness of these solutions. Once students learn to apply graphs to represent the information in problems, they can potentially learn to apply algorithms to solve problems.

DISCUSSION

Reflections across the three lesson studies revealed limitations and concerns, informed our research approach, and identified future research questions and studies. The following sections will address these reflections.

Limitations and Concerns

We have been able to draw a few ongoing lessons from these early explorations into interdisciplinary collaboration. For example, when working with teachers and students in actual classrooms, the attrition rate is much larger than anticipated, which likely stems from demands on teachers' time and the overcrowded school day. The addition of other researchers to assist in the classroom would eliminate some of the factors that lowered the overall numbers. Further, research ethics requirements have increased the demands on researchers and may have negative effects on students and parents being willing to participate in research studies. Clearly, classroom-based research requires collaboration and trust amongst researchers, teachers, students, and parents (Anthony et al., 2009). We believe that to broaden these interdisciplinary collaborations from Education and Computer Science into other areas (e.g., psychology and applied statistics) may prove fruitful and improve the validity of this research.

Computer Science Education

An understanding of technology, the role it plays in society, and how to use it responsibly are important components of education today. In addition to supporting learning in general, CS students learn a number of important skills, including problem solving, algorithmic thinking, and logical reasoning (CSTA, 2006). The integration of CS topics into mathematics classes has the potential to support a number of processes; for example, visual abstractions common in computer science can support the communication process; logical reasoning is a fundamental component of the reasoning process; and problem solving, an integral CS skill, is fundamental to many aspects of learning. CS is a broad subject, comprised of many different specialized areas of study; a key step in evaluating its teachability in elementary classrooms is determining appropriate CS topics. Niman (1975) identified graph theory as a potential topic for elementary school instruction due to its versatility in visually representing ideas and its application to puzzles.

Just as important as an understanding of technology is ensuring that future CSEd research has a sound theoretical and practical foundation. If existing publications are indicative of the current state of CSEd research, then researchers in this area should select studies upon which to base their work with caution. Randolph, Julnes, Sulinen, and Lehman (2008) noted a number of shortcomings and flaws found in CSEd research publications, including flaws in reporting the elements recommended by the American Psychological Association, problems with sampling of participants and self-selection, and a lack of validity and reliability of measures. They also noted that many of the studies analyzed used research designs that suffer from weak internal validity.

With this constructive criticism in mind, our research group chose a collaborative approach in designing and deploying research studies. We addressed some of these concerns about individual studies by a planned research agenda that evolves from individual lesson studies that explore appropriateness, to sequential lesson studies that develop valid and reliable measures and documentation procedures, and further to experimental designs that investigate cause–effect relationships using proper interpretation techniques for the data. Recognizing that as computer scientists we are not experts in the field of human research, we have made connections and found collaborators in the field of education research. Equally important, we keep in mind that human research studies almost invariably fail to conform exactly to theoretical plans (i.e., working within the educational system with students and teachers can prove 'messy'). Faced with improperly implemented testing procedures and participant attrition, researchers ought to be flexible and need to be adaptable. Ultimately, just because a study deviated from the intended design, this does not mean that no valuable information can result.

As an example of this approach, we cite our recent study on the impact of graph theory instruction in Grade 6 mathematics classes (Carruthers, 2010a). We chose to use a contingency table analysis for the nominal and ordinal data (i.e., the quality of a graph drawn by participants in the experimental groups to which they were assigned). While perhaps less well known than analysis techniques for continuous (i.e., ratio or interval) data, contingency table analysis can be a useful way to investigate possible associations between variables (e.g., whether or not a graph was drawn pre- and postintervention) as well as associations between an intervention or outside influence and variables. Categorical data are quite common in educational research and these data can be either ordinal or nominal, with each type requiring different analysis techniques.

For our study, the resulting data were ordinal in nature; that is, there was a sense of order to the values for the type of picture that participants drew as part of their solution to the problems: 0 for no picture, 1 for a nongraph picture, and 2 for a graph (vertices and edges). As the intervention taught how to use graphs to solve problems, drawing a graph was considered better than a nongraph picture that in turn was better than no picture at all. We were interested in whether or not there might be a higher incidence of the use of graphs following graph theory instruction. This type of association can be evaluated using McNemar's test of association (Elliot & Woodward, 2007). Since we are essentially measuring two separate readings of the same characteristic, it is expected that the measurements would be similar to some extent; therefore, it is not necessarily meaningful to determine if the variables are independent. However, if there is an outside force that might influence one measurement more than the other (in this case, an intervention), then there might be a shift in the counts. The McNemar test of association can indicate whether the observed shift is significantly different from the probability of no association.

This analysis technique usually requires minimum cell counts of 5; unfortunately, our small data set ($N = 9$) rendered it unusable. Reduced participation rates, particularly in the control groups (those not engaged in the intervention lessons on graph theory), combined with improperly administered posttests resulted in descriptive data tables with much reduced cell counts in the final data. In some cases, data sets were limited to less than 10 data points total; so it was impossible to expect a minimum cell count of 5 in a 2x2 or 3x3 matrix, which precluded the intended statistical analysis techniques. This meant that high or low *p*-values associated with the measures were suspect and no measure of significance could be given. Significance indicates the probability of test statistics occurring by chance. Normally, a significance value greater than .05 would suggest rejection of the experimental hypothesis; however, this does not mean that the null hypothesis is true. The null hypothesis is one of no effect; however, all a nonsignificant result indicates is that the effect is not big enough to be anything other than a chance finding—it does not indicate that the effect is zero. Cohen (1990) pointed out that a nonsignificant result should never be—despite that it often is—interpreted as no relationship between variables.

Rather than being deterred by the fact that we were unable to claim significance, we chose instead to treat the results as exploratory in nature. Trends and patterns that emerged from the data, while not necessarily significant, could still be indicative of a possible effect and hint at places that may be worthy of further investigation. What is essential is to carefully account for and report any deviations from the intended research methodology and analysis procedures. Readers of research studies need to be informed of the reasons for post hoc changes in analysis or methodology, and future studies should be able to learn from the mistakes of others in order to improve practices and refine research designs. We believe that adherence to these practices by both ourselves and other researchers in the field will contribute to addressing concerns around internal validity.

Future Research

It is widely assumed that collaboration in research is desirable and that it should be encouraged (Katz & Martin, 1997). Canada, through its Natural Sciences and Engineering Research Council, has demonstrated interest in the notion of research collaboration via Pacific CRYSTAL, of which all the contributors to this book are a part. Katz and Martin (1997) identified several factors that motivate collaboration: funding agencies' need to save money, the growing availability and falling (real) cost of transport and communication, the desire for intellectual interactions with other scientists, the need for a division of labour in more specialized or capital-intensive areas of science, the requirements of interdisciplinary research, and government encouragement of international and cross-sectoral collaboration. Many of these reasons motivated our research through Pacific CRYSTAL in general and the Education/Computer Science collaboration specifically.

Educational research can be generally defined as the formal, systematic application of the appropriate scientific method (i.e., quantitative, qualitative, mixed method) to the study of educational problems. Educational research findings are important because they shape and influence teacher education programs and help define the intellectual context within which all involved in education work. However, there is criticism of the quality of research in the field (i.e., its partisan nature, methodological shortcomings, nonempirical approaches, 'great thinkers' adulation, and lack of relevance to practice, policy, or theoretical approaches) serious enough to raise both concerns and misgivings (Tooley & Darby, 1998). A major advantage to our collaboration across disciplines is that some, if not all, of these concerns and misgivings have been addressed.

With the three pilot studies reported in this chapter, we have restarted an investigation on the possibility of teaching basic CS concepts earlier in students' schooling. Our future research plans include expanding on the studies to test the preliminary but promising findings for the three concepts as well as increasing the list of core concepts and problem-solving techniques applicable to elementary, middle, and secondary school students. Besides understanding what concept or technique can be taught in which grade and at what level, we are interested in long-term consequences of CSEd; that is, does teaching CS concepts early address the mainline goal for citizens and the pipeline goal of CS students and careers? What is the best time to start with CSEd and at what level? Does CSEd improve students' decision making regarding career choices? Does CSEd improve the quality of CS students? Does CSEd increase the number of CS students in postsecondary institutions to a level that satisfies the supply–demand ratio? Does CSEd improve general problem-solving capabilities? Does CSEd ensure more responsible use of social networking tools on the Internet?

Of course, the path to understanding the long-term consequences is not easy. How can we best realize the teaching of CSEd in K–12? We believe that an interdisciplinary and cross-disciplinary approach is the most promising and is possible in a rather seamless way. CS activities include many prescribed learning outcomes from K–12 curricula (see a discussion on this topic by Romanow et al., 2008, for BC) from basically all disciplines. Many activities, if prepared carefully, are easily approachable (a great example is the ongoing *Computer Science Unplugged* by Bell et al., 2006) not just for the students but also for the CS-inexperienced teacher. Commitment from governments and school districts for including CSEd as a priority into curricula could make this possible (see BC's environmental education guide, http://www.bced.gov.bc.ca/environment_ed/). We believe that a better education in CS in K–12 does not just educate better citizens (mainline goal) and increase the number of students in CS programs at universities and colleges—we believe it will attract a student population better suited to the discipline (pipeline goal).

REFERENCES

Anthony, R. J., Yore, L. D., Coll, R. K., Dillon, J., Chiu, M.-H., Fakudze, C., et al. (2009). Research ethics boards and gold standard(s) in literacy and science education research. In M. C. Shelley II, L. D. Yore, & B. Hand (Eds.), *Quality research in literacy and science education: International perspectives and gold standards* (pp. 511–557). Dordrecht, The Netherlands: Springer.

Bell, T., Witten, I. H., Fellows, M., Adams, R., & McKenzie, J. (2006). *Computer science unplugged. An enrichment and extension programme for primary-aged children* (teacher ed.). Retrieved from http://www.csunplugged.org/

Briggs, J. (1992). *Fractals: The patterns of chaos*. New York: Touchstone.

Carruthers, S. (2010a). *Grasping graphs*. Master's thesis, University of Victoria. Retrieved from http://hdl.handle.net/1828/3193

Carruthers, S. (2010b). *An interdisciplinary guide for K–8 computer science (CS) education* [Poster]. Presented at the 41st ACM Technical Symposium on Computer Science Education (SIGCSE'10), Milwaukee, WI, USA.

Carruthers, S. (2010c). *Relational graphs: What are they?* [DVD]. Presented at the 41st ACM Technical Symposium on Computer Science Education (SIGCSE'10), Milwaukee, WI, USA.

Carruthers, S., Milford, T. M., Pelton, T. W., & Stege, U. (2010). Moving K–7 computer science instruction into the information age. In *Proceedings of the 15th Western Canadian Conference for Computing Education (WCCCE'10)* (pp. 1–5).

Carter, L. (2006). Why students with an apparent aptitude for computer science don't choose to major in computer science. In *Proceedings of the 37th SIGCSE Technical Symposium on Computer Science Education (SIGCSE'06)* (pp. 27–31).

Cohen, J. (1990). Things I have learned (so far). *American Psychologist*, *45*(12), 1304–1321.

Dijkstra, E. W. (n.d.). Retrieved from Wikipedia, the Free Encyclopedia: http://en.wikipedia.org/wiki/Edsger_W._Dijkstra

Elliot, A. C., & Woodward, W. A. (2007). *Statistical analysis quick reference guide with SPSS examples*. Thousand Oaks, CA: Sage.

Fincher, S., & Petre, M. (2004). *Computer science education research*. London, England: Routledge Falmer.

Flakener, N., Sooriamurthi, R., & Michalewicz, Z. (2010). Puzzle based learning for engineering and computer science. *IEEE Computer Society*, *43*(4), 20–28.

Ford, G. (1982). A framework for teaching recursion. *SIGCSE Bulletin*, *14*(2), 32–39.

Goode, J. (2008). Increasing diversity in K–12 computer science: Strategies from the field. In *Proceedings of the 39th SIGCSE Technical Symposium on Computer Science Education (SIGCSE'08)* (pp. 362–366).

Goodrich, M. T., & Tamassia, R. (2002). *Algorithm design: Foundations, analysis and internet examples*. New York: Wiley.

Grimaldi, R. P. (1998). *Discrete mathematics* (4th ed.). Reading, MA: Addison-Wesley.

Gunion, K. (2009). *FUNdamentals of CS: Designing and evaluating computer science activities for kids*. Master's thesis, University of Victoria. Retrieved from http://hdl.handle.net/1828/2750

Gunion, K., Milford, T. M., & Stege, U. (2009a). Curing recursion aversion. In *Proceedings of the 14th ACM SIGCSE Conference on Innovation and Technology in Computer Science Education (ItiCSE'09)* (pp. 124–128).

Gunion, K., Milford, T. M., & Stege, U. (2009b). The paradigm recursion. *Journal of Problem Solving*, *2*(2), 142–172.

Haberman, B., & Averbuch, H. (2002). The case of base cases: Why are they so difficult to recognize? Student difficulties with recursion. *SIGCSE Bulletin*, *34*(3), 84–88.

Hsin, W. (2008). Teaching recursion using recursion graphs. *Journal of Computing Sciences in Colleges*, *23*(4), 217–222.

Katz, J. S., & Martin, B. R. (1997). What is research collaboration? *Research Policy*, *26*(1), 1–18.

Klein, A. (2006). K–12 education shrinking future college graduate population in computer studies. *Journal of Computing Sciences in Colleges*, *21*(4), 32–34.

MicroWorlds EX [Computer software]. (2010). Retrieved from http://www.microworlds.com/solutions/mwex.html

Niman, J. (1975). Graph theory in the elementary school. *Educational Studies in Mathematics*, *6*(2), 351–373.

Ormond, J. E., Saklofske, D. H., Schwean, V. L., Harrison, G. L., & Andrews, J. J. W. (2006). *Principles of educational psychology* (2nd Canadian ed.). Toronto, ON, Canada: Pearson Education.

Randolph, J., Julnes, G., Sulinen, E., & Lehman S. (2008). A methodological review of computer science education research. *Journal of Information Technology Education*, *7*(1), 135–162.

Romanow, H., Stege, U., Agah St Pierre, A., & Ross, L. (2008). *Increasing accessibility: Teaching children important computer science concepts without sacrificing conventional subjects of study*. Presented at the 13th Western Canadian Conference on Computing Education (WCCCE'08). Retrieved from http://outreach.cs.uvic.ca/2008WCCCE.pdf

Schunk, D. H. (2000). *Learning theories: An educational perspective* (3rd ed.). Upper Saddle River, NJ: Merrill.

Service Canada. (2011). *Computer programmers and interactive media developers*. Retrieved from http://www.servicecanada.gc.ca/eng/qc/job_futures/statistics/2174.shtml

Slonim, J., Scully, S., & McAllister, M. (2008). Crossroads for Canadian CS enrollment: What should be done to reverse falling CS enrollment in the Canadian education system? *Communications of the ACM*, *51*(10), 66–70.

Tooley, J., & Darby, D. (1998). *Educational research - A critique*. London, England: Office for Standards in Education. Retrieved from http://www.ofsted.gov.uk/Ofsted-home/Publications-and-research/Browse-all-by/Education/Leadership/Governance/Educational-research-a-critique-the-Tooley-report

United States Computer Science Teachers Association. (2006). *A model curriculum for K–12 computer science: Final report of the ACM task force curriculum committee* (2nd ed.). New York: Author.

Zweben, S. (2008). *Computing degree and enrolment trends from the 2007–2008 CRA Taulbee Survey: Undergraduate enrolment in computer science trends higher; doctoral production continues at peak levels*. Washington, DC: Computing Research Association. Retrieved from http://archive.cra.org/taulbee/CRATaulbeeReport-StudentEnrollment-07-08.pdf

Sarah Carruthers, Yvonne Coady, Celina Gibbs, Katherine Gunion and Ulrike Stege
Department of Computer Science

Todd M. Milford
Department of Educational Psychology
University of Victoria
Victoria, British Columbia, Canada