We report on a mathematician’s perceptions and awarenesses related to incorporating problem-based activities requiring computational thinking into an upper level undergraduate mathematics course. Computational thinking is understood as the thinking, strategies, and approaches for problem solving that parallel the design of computational algorithms which can be followed and executed by a computer. Data from this case study is qualitative in nature, and seeks to present an in-depth account of one professor’s experiences developing and teaching computational thinking in and for mathematics. Analyses highlight the similarities and differences amongst the values and opportunities perceived for computational thinking versus other more ubiquitous mathematical approaches, as well as the perceived tensions and challenges in trying to foster such values and opportunities.

Key words: Computational Thinking; Undergraduate Mathematics; Awarenesses; Disciplinary Goals; Mathematics Professor

This study reports on the case of a mathematician’s aims and objectives when incorporating a focus on computational thinking in an upper level undergraduate course focused on problem solving, simulations, and mathematical investigation. Computational thinking can be loosely defined as the thinking involved in, and related to, computer programming. This can include screen-based programming, paper-based or embodied pseudo-coding, as well as other approaches to problem solving that parallel the design of computational algorithms which could be followed by a computer (Aho, 2012; Wing 2006). While the idea of using computer programming in mathematics is not new (e.g., Feurzeig, et al., 1969; Howson & Kahane, 1986; King et al., 2001; Marshall et al., 2014a; Papert, 1980), research has been primarily focused on elementary or secondary school learning (e.g., Floyd et al., 2015; Gadanidis, 2014, 2015; Sneider et al., 2014) with recent attention turning towards ways computer programming can be used in undergraduate mathematics (Muller, et al., 2009; Marshall & Buteau, 2014). However, purposes and best practices for computational thinking in undergraduate mathematics learning are far from well understood.

This research is part of a broader research program that aims to address the what, why, and how of computational thinking in and for undergraduate mathematics learning. In this paper, we focus specifically on the experiences and perceptions of a mathematician in his design and implementation of a mathematics course that sought to incorporate computational thinking as a key practice. In particular, we address the following research questions:

1. What does a mathematician teaching an undergraduate level problem solving course describe as the values and opportunities for student learning when incorporating computational thinking in and for mathematics?

2. What challenges and tensions were experienced from a teaching perspective?

We use a case study approach and qualitative analysis, which are suitable for collecting and interpreting in-depth stories of teaching and learning (e.g., Stake, 2000; Yin, 1994). We use Mason’s (1998) framework of levels of awareness to analyze both the disciplinary and pedagogical values associated with computational thinking in and for mathematics, as well as our participant’s perceptions of opportunities, challenges, and tensions.
Background

Research has highlighted an extensive set of skills and competencies that may be developed by incorporating computational thinking in mathematics learning. These include, but are not limited to: self-motivation to do and explore mathematics; experimentation; development of mathematical intuitions and approaches; critical reflection; working with abstraction and different representations (Howson & Kahan, 1986; King et al., 2001; Marshall & Buteau, 2014; Marshall et al., 2014). Addressing mathematics problems through the use of computer programming has also been described as transformative to learning (Papert, 1980), fostering learner-driven active engagement, as well as resilience and creativity in the face of new and challenging problems.

Parallels between learning computer programming in general, and developing computational thinking for mathematics in particular, have been noted. Wing (2008) observes that: “In computing, we abstract notions beyond the physical dimensions of time and space. Our abstractions are extremely general because they are symbolic, where numeric abstractions are just a special case” (p.3717). In considering higher levels of mathematics, abstractions become more complex than simply numeric abstractions and research has suggested that the more condensed an abstract object is, the more challenging it is to reason with (e.g., Hazzan, 1999). For example, operating on a set of numbers is seen as a more abstract, and more complex, endeavor than operating on a single number. As such, computer programming may offer a sort of scaffold in fostering abstract reasoning by increasing students’ experiences with abstraction while providing tangible and immediate feedback on how such abstractions may be operated upon.

Weintrop et al. (2016) developed their taxonomy for computational thinking in mathematics and science by analyzing characteristic practices that were seen as the most important in meeting both the needs of students and the disciplinary practices of mathematics and science professionals. They cite their work as contributing a set of actionable guidelines for bringing computational thinking into mathematics and science classrooms quickly and effectively that can “serve as resource to address “what” and “how” questions that accompany the creation of new educational materials” (p.129). Their taxonomy is depicted in Figure 1, with some elaboration of practices below.

![Figure 1: Computational thinking in mathematics and science taxonomy (Weintrop et al., 2016, p. 135)](image-url)
As the professor participating in our study focused his course on problem solving and simulations, we take a moment to explicate some of the facets in the two corresponding dimensions of the taxonomy. **Modeling and Simulation Practices** focus on skills in using, analyzing, designing, and constructing computational models. Models may be used “to test many different solutions quickly, easily, and inexpensively” (p.137), and understanding how the model relates to the phenomenon being represented can help students articulate the similarities and differences between the two. Noticing similarities and differences is part and parcel to rich mathematical thinking (Mason, et al., 1982), and in the context of computational thinking can be fostered by questions such as “what assumptions have the creators of the model made about the world and how do those assumptions affect its behavior?” and “what layers of abstraction have been built into the model itself and how do these abstractions shape the fidelity of the model?” (Weintrop, et al, 2016, p.137). **Computational Problem Solving Practices** include interpreting and preparing problems for modeling, choosing between and assessing different tools and approaches, developing solutions, and debugging or revising. Students must be aware of which problems can be effectively addressed through computational solutions as well as how to reframe a problem so that “existing computational tools – be they physical devices or software packages – can be utilized” (p.138). Choosing amongst possible strategies, tools, or solutions, and developing modular approaches can foster critical mathematical thinking and logical deduction, while equipping students with a set of approaches that can be applied to larger, more complex problems. Of interest in this research study is whether and which practices emphasized in the taxonomy are recognized, valued, and incorporated by the professor teaching a problem-based mathematics course, and which practices may be seen as more challenging to incorporate.

**Theoretical Framework**

Mason’s (1998) article discusses the necessary levels of awareness which distinguish between a novice, an expert, and a teacher in the discipline of mathematics. These levels of awareness are linked to the structure of attention, which “encompasses the locus, focus, and form of attention moment by moment” (p.250). Through shifts of attention, awareness is broadened and an individual may move from novice, to expert, to teacher through his or her attention and sensitivity toward different aspects of the discipline. Mason (1998, p.256) identifies and develops three forms of awareness:

- **Awareness-in-action**: this is the awareness which focuses attention on what to do in the moment. It is “highly personal and context specific” (p.256) and enables us to act, to know what to do in response to some stimulus. Awareness-in-action includes the “power to select, distinguish, demarcate, discern, detect differences; ... to see (construct) something as an example of something else... to abstract... to connect... to express... [and] to decide” (p.257).

- **Awareness-in-discipline**: this awareness is awakened when one becomes aware of his or her awareness-in-action. In other words, it is an awareness “which enables articulation and formalisation of awarenesses-in-action” (p.256). Awareness-in-discipline is broadened by the ability to formalize algebra and geometry, to appreciate how and why we act in the moment, to enact the “habits of thought, forms of fruitful questions, and methods of resolution of those questions” (p.259) applicable to the discipline of mathematics.

- **Awareness-in-counsel**: this is the awareness required of teachers. It is an awareness of awareness-in-discipline; the “self-awareness required in order to be sensitive to
what others require in order to build their own awareness-in-action and –indiscipline” (p.256). Awareness-in-counsel “provides access to sensitivities which enable us to be distanced from the act of directing the actions of others, in order to provoke them into becoming aware of their own awarenesses” (p.261).

In Mason’s perspective (and we agree), teaching is not about telling students what to do, but rather it involves eliciting and fostering their ability to decide for themselves what to do. In the context of undergraduate mathematics education, an instructor’s awareness-in-counsel is evidenced in his or her choices (and the expressed reasons behind these choices) of what content, practices, habits of minds to emphasize, as well as in the features of the problems or tasks chosen to elicit and support the shifts of attention that will help novice become expert.

As an analytic lens, we use the construct of awareness to interpret and analyse a mathematician’s decisions and reflections regarding the values, opportunities, challenges, and tensions of teaching a problem-solving mathematics course with a focus on computational thinking. We focus in particular on our participant’s awarenesses related to the taxonomy of computational thinking and practices developed by Weintrop et al. (2016) as they are associated with, or different from, mathematical thinking and practices more generally. In particular, we examine what our participant identified as valuable ways of knowing and practicing mathematics with computational applications, what opportunities were envisioned or provided, what challenges emerged, and what tensions were felt in terms of realizing those values and opportunities in the face of the emergent teaching and learning challenges.

Methodology

Our participant was an experienced and well-received mathematics instructor, whom we shall refer to as Dr. Y. Dr. Y has taught at the undergraduate level for over 25 years, and this was his third time teaching a course which emphasized computational thinking for mathematics problem solving. Since this research focuses on an in-depth qualitative analysis of an individual’s experiences and decisions regarding teaching and learning, a case study approach (e.g., Stake, 2000; Yin, 1994) is appropriate. Data include field notes taken during the course by one of the researchers, personal reflections and instructional materials provided by Dr. Y, as well as unstructured interviews conducted at the completion of the course.

The course taught by Dr. Y was an undergraduate third year problem solving course for mathematics majors. The main objectives of the course included enriching students’ existing content-based knowledge and enhancing their problem solving strategies and thinking. Classes were conducted in 50-minute blocks, and students spent the majority of class time working on problems, which they could choose to do individually or in small groups. The problems were chosen to elicit or provide different perspectives on material that was covered in other courses, as well as to stimulate reflection on various new and familiar ways of engaging in solving problems. Students were expected to discuss and justify their solutions with their peers and their instructor, and were required to present solutions to the class at various times throughout the semester. Weekly assignments were assessed, as well as two in-class tests and a final exam. As part of his description of the course, Dr. Y emphasized (and this was confirmed via field notes) that very little of class time was spent on lecturing. Dr. Y reported that lecturing was “done only to introduce a problem or a set of related problems, and provide context and motivation. Students realize that the problems are not in any way ‘randomly’ selected, but instead are tied to their previous knowledge and experiences.”

Students in the course were described as “mathematically mature” – each had taken several second year courses, with experience working with theorems and proofs, and had “developed a critical attitude toward the level of rigour required of a proof”. However, students’ computer programming background varied widely: some had no programming
experience beyond using one-line commands in Maple or Matlab, while others had taken computer science courses and/or had written programs in several languages. Dr. Y indicated that as of Fall 2016 this would change, as students will be required to complete a new programming requirement by the end of their second year. In this problem solving course, student learning objectives included “practicing math the way math researchers do” and “learning to write programs in Python and use them to investigate problems in math”. At the time of this research, Dr. Y indicated that approximately 40% of class time and assessment was devoted to computational thinking, with an emphasis on the “basic building blocks” of coding such as conditional statements and loops. Detailed course objectives were co-constructed throughout the course as students discovered and developed an extensive list of strategies for problem solving.

Results and Analyses

We organize this section around our two research questions, addressing respectively:
1. Values and opportunities: we examine Dr. Y’s perceptions via his choices in pedagogical structures, such as features and affordances of exemplar tasks, as well as his expressed motivations and intentions for course activities and objectives;
2. Challenges and tensions: we rely on field notes, reflections, and interview data to analyse Dr. Y’s experiences enacting his teaching agenda within institutional constraints and in response to student feedback.

Values and opportunities

As previously mentioned, Dr. Y spent minimal class time lecturing students and the value he placed on providing opportunities for students to “discover” and “develop” were made explicit in course objectives. Exemplifying his awareness-in-counsel, he sought to foster students’ powers in selecting and distinguishing problem-solving strategies (awareness-in-action) and to compel them to formalize this awareness via their articulation of (e.g.) which kinds of problems could be reformulated in such a way as to be solvable with programming (awareness-in-discipline). To this end, students encountered a variety of problems and tasks which required reformulation, and in the cases where the problems could be fruitfully solved through programming, students were required to use, assess, and design computational models for solving – features of the modeling & simulation practices described by Weintrop et al. (2016). One of the early examples of a programming activity to which students were exposed introduced the concept of a two-dimensional random walk and required students to construct a program which simulated $N$ steps of the walk (start at the origin, and with equal chance move up, down, left, or right).

Dr. Y described some of his reasons for using this problem:
“The problem itself is not so difficult. Most students have little or no problems writing and running the code. However, testing the code is a challenge, because there is a new element in it – randomness. Students are familiar with testing a ‘deterministic’ program, where they could predict the answer, run the program and see whether or not their answer agrees with the computer’s answer. But how does one test a program whose output is based on randomness? This is more difficult. Visualization helps, trial and error, tinkering with the code… Eventually students have to figure out the distance between the initial and terminal points of a random walk with $N$ steps, and then compute the average distance as the random walk is repeated many times. Now they can compare the average distance they obtained to a theoretical result, thus having a way of checking that their code is correct.”
In this excerpt, we identify several points of interest. First we note Dr. Y’s use of a “not so difficult” problem that increased in complexity through the practice of testing and verifying the code. Following Papert’s (1980) lead, such a “low-floor / high-ceiling” approach requires minimal prerequisite knowledge yet offers opportunities to investigate more complex ideas, practices, and relationships. Referring to Weintrop et al.’s (2016) taxonomy, we note opportunities for students to develop computational problem solving practices, such as preparing problems for computational solutions (“writing and running the code”), assessing different approaches / solutions (“testing the code”), and troubleshooting and debugging (“tinkering with the code”). We interpret the selection of this problem as an instantiation of Dr. Y’s awareness-in-counsel, namely, the sensitivities required to direct novice students’ attention toward complex computational practices required for disciplinary expertise. A second point of interest is Dr. Y’s acknowledgement of the complexities associated with random systems, and the role of visualization in helping to attain a theoretical result. Randomness can be challenging for mathematics learners to address, and novice strategies for dealing with randomness have included various ways of reducing its level of abstraction (e.g., Chernoff & Mamolo, 2015). By contextualizing questions related to randomness within computational thinking, students may be provided with scaffolding for developing abstract reasoning – the tangible feedback given by the program they construct, test and adapt offers insight that may not be readily perceived in other contexts. Similarly, the insights gained through the practice of visualization have been linked to conceptual development in higher mathematics (e.g., Tall, 2007), as well as achievement and understanding (e.g., Koedinger, 1992). A final point we note is the connection, within this task, of mathematical disciplinary practices and content and computational practices. The computational practices support mathematical ones and facilitate the development of mathematical content, while the act of verifying that students’ code is correct highlights an important difference between the two disciplines and their practices – specifically, as Dr. Y put it, that “providing evidence that a computer code is correct is different than a formal math proof!”

Challenges and Tensions

With respect to institutional constraints, some challenges were already mentioned – the diversity in students’ prior programming experience, the scheduling of three short classes per week, as well as the size of the course. The course runs with 30-40 students in a regular classroom (rather than a computer lab), and students are required to bring their own portable devices in order to engage with class activities – while most students brought notebook computers, some tried to work on their phones and others had no devices. A few of the observed challenges included syntactical differences in Python depending on what operating system was in use – for instance, the tab function will work for indenting in iOS but yields an error in Windows. While not a major concern, such discrepancies needed to be negotiated during students’ peer-peer interactions and work. A more significant issue that emerged occurred when students either did not own, or did not bring, their own portable computer. This led to group work that positioned some students at the periphery, while others were central to the activity. Dr. Y expressed frustration at this imbalance, and noted that “students who watched someone else type and work on screen seemed to learn less, and were less confident in their coding.” This was a particular challenge during problem-solving activities wherein students would go straight to on-screen programming solutions, without preliminary discussion or paper-work, making it difficult for other students (and the instructor) to follow their thinking. Dr. Y also noted that the 50-minute classes were not ideal for his pedagogical approach. The time limit constrained the types of problems selected for the course, stemmed peer-peer discussions, and inconvenienced students – Dr. Y remarked: “just as students get in
the ‘zone’ [problem-solving zone] tackling some rich mathematics, the class ends and they’re off to biology or physics or something completely different.” Watson (2008) has touted the importance of extended experiences with mathematics, noting that it is more difficult to learn the subject when attention is constantly shifting from one task to another. We see this as particularly relevant for computational thinking in mathematics problem solving, as the added need to reformulate a question in programming terms and to coordinate theoretical and computational solutions may require additional time, especially for inexperienced learners. In reflecting on these issues, Dr. Y’s attention seemed to shift from acknowledging the constraints to posing questions around how to better navigate them, suggesting a broadening of his awareness-in-counsel. He raised questions around how to construct appropriate activities that include programming and which could be meaningfully tackled given the aforementioned constraints of time, class-size, and student access to computing devices.

A notable tension that emerged for Dr. Y concerned the use of group work and how this impacted ways in which he was able to fairly assess students. He noted that previous offerings of the course had relied primarily on take-home assignments that could be completed in groups, but were to be submitted individually. Dr. Y noted “in a problem-solving course, it makes sense to assess problem-solving and it’s difficult to do that in a midterm or exam.” However, concerns emerged: “most students were handing in perfect, or near-perfect, solutions, often very similar to their peers’ solutions, and yet different from the kinds of work they were producing and presenting in class. It was obvious that they did not take the course requirements seriously.” In addition, Dr. Y identified challenges in assessing programming-based problem solving as “when students work on-screen, usually overwriting the code, it’s hard to follow their thinking.” As a result, Dr. Y introduced a significant in-class assessment component comprised of tests and exam, and accounting for 80% of the course grade. Dr. Y lamented, “I’m not sure what else to do.”

Concluding Remarks

Despite the tensions and challenges which emerged, Dr. Y’s perceptions of the role of computational thinking in and for undergraduate mathematics were positive. Dr. Y reflected: “Very often, when we teach math, we ‘forget’ how results, theorems, or definitions have been arrived at. We present a proof, but rarely talk about how it was constructed, and in particular, we don’t talk about failed attempts! By attempting to solve a problem, students experience some of that essential process of creating mathematics. By writing a computer program, they can engage in rich mathematics, develop important habits of mind, and produce something tangible in the end”

Dr. Y plans to increase the emphasis on computational thinking in future instances of the course, and we plan to extend this research following him through changes inspired by the shifts of attention and broadening of awarenesses that occurred during the course of this study. With a growing emphasis on computational applications of mathematics, both in industry and education, there is a need to better understand the role that mathematics courses can play in fostering computational thinking. Attention toward how the broad array of computational practices for mathematics may be developed is needed, particularly for those practices which closely relate to the mathematical habits of mind necessary for disciplinary expertise. Yadav et al. (2014) noted that fewer than 15% of the pre-service teachers surveyed recognized critical thinking as part of computational thinking, and fewer than 10% viewed computational thinking as helpful in understanding the “why” behind problems, suggesting work is needed to raise awareness of what is computational thinking (in and of itself, as well as in connection to mathematics) and what are fruitful learning experiences for students.
References


