Int. J. Res. Undergrad. Math. Ed.
https://doi.org/10.1007/s40753-017-0066-1

Levels of Programming in Mathematical Research and University Mathematics Education

Laura Broley[1] · **France Caron**[2] · **Yvan Saint-Aubin**[3]

Abstract In response to a recent Canada-wide survey, where it was found that mathematicians use computer programming much less in their teaching than in their research, an exploratory study involving 14 Canadian mathematicians sought to gain a deeper understanding of the place of programming in both contexts. To capture the differences in the degree of interaction with programming in various mathematical practices, a scale of six levels has been defined and used. While providing visibility to an aspect of mathematical practice that is often absent from published work, the views of our participants also highlight some important issues that would require attention in order to reduce the identified gap, should that be deemed the favourable direction to take.

Résumé En réponse à un sondage pancanadien récent, qui révélait que les mathématiciens utilisent beaucoup moins la programmation informatique dans leur enseignement que dans leur recherche, une étude exploratoire auprès de quatorze mathématiciens canadiens a cherché à mieux comprendre la place de la programmation dans ces deux contextes. Une échelle à six niveaux a été définie et utilisée pour caractériser le degré d'interaction avec la programmation dans différentes pratiques mathématiques. Tout en rendant visible un aspect de la pratique mathématique souvent

✉ Laura Broley
l_brole@live.concordia.ca

[1] Department of Mathematics and Statistics, Concordia University, Montréal, Canada

[2] Faculté des Sciences de L'éducation, Département de Didactique, Université de Montréal, Montréal, Canada

[3] Département de Mathématiques et de Statistique, Université de Montréal, Montréal, Canada

Published online: 30 November 2017

absent des publications, le témoignage des participants à l'étude met en évidence certaines contraintes qu'il conviendrait de revoir si l'on souhaitait réduire l'écart entre les pratiques de recherche et celles qui sont développées dans la formation.

Keywords Mathematical practices · Mathematicians · Undergraduate teaching and learning · Computer programming · Institutional constraints

Introduction

In this so-called "digital era", where one can graph a function or compute an integral simply by speaking into a mobile phone, it may be hard to remember that only 40 years ago, the power of computer technology to perform even these simple tasks was accessed only through "programming".[1] Today's digital landscape includes a diverse set of software and applications capable of supporting all sorts of mathematical activity "at the press of a button"; and current discussions about using the computer in the teaching and learning of mathematics reflect this diversity. Nonetheless, programming has remained a significant part of science, technology, engineering, and mathematics (i.e., STEM) related disciplines, where professional work often involves not just the use of existing digital tools, but also the creation of new and/or more adequate computer programs. This is just one reason why programming has recently regained the attention of politicians, curriculum developers, and researchers worldwide, who envision it as having potential both within and beyond classrooms.

But one cannot forget that programming has had a relatively long history in mathematics education, generating as much enthusiasm as resistance. It therefore seems important to learn more about the current scope, nature, and conditions of programming integration in the teaching and learning of mathematics. Given the mutual evolution of computer technology and human activity, it also seems useful to better understand the place of programming in the work of today's mathematician.

Universities present a rich context for studying these topics, as they are both hubs of mathematical research in a variety of fields and educational institutions offering courses taught by research mathematicians. We might assume that with easier access to technology and greater liberty, university professors could integrate authentic programming activities into their teaching more easily than their school counterparts. Yet, a national survey brings forth some doubts: while 43% of 302 Canadian mathematicians reported using programming in their research, only 18% claimed to use it in their teaching (Buteau et al. 2014). We wanted to know why such a large gap existed; and that required us to examine more closely the specific ways in which mathematicians develop and use computer programs in their research, as well as the types of programming activities they make available to students in their courses.

The present paper reflects on an exploratory study involving 14 mathematicians (Broley 2015), which sought to uncover some of the story behind the statistics. Building on a presentation of its results (Broley 2016), we begin by placing the study in the context of two classical topics: computer programming in mathematics

[1] The use of quotations here indicates a lack of clarity in the definition of programming. We return to this later. Readers, who are likely to have their own implicit definition, should be aware of a potential ambiguity.

education, and the gap between mathematical activity in educational institutions and the research practices of professional mathematicians (thereafter professional mathematical practices). We then summarize elements of Chevallard's (1998) Anthropological Theory of the Didactic (thereafter ATD), which were used in carrying out the study. We also add to our theoretical framework some notions developed in the context of professional practices in education (Morrissette 2011). The perspectives of our participants are then presented, with the aim of describing, comparing, and explaining some elements of "programming" "use" in mathematical research and university mathematics education. We conclude by discussing the variable status of programming within the mathematics community and university mathematics departments.

Context and Theoretical Underpinnings

The Rise, Fall, and Revival of Programming in Mathematics Education

As personal computers became available in the 1970s and 80s, scholars began reporting on the potential of programming for exploring mathematical ideas or developing ways of thinking (cf. Papert 1980; Knuth 1985; Leron and Dubinsky 1995). Despite the initial enthusiasm, however, there were relatively few long-lived classroom implementations and little impact on curricula. The disappearance of programming from mathematics education literature followed throughout the 1990s (Lagrange and Rogalski 2015). Computer science remained a self-contained discipline, and educators began focussing on the user-friendly powerful tools developed for mathematical purposes, such as spreadsheets, dynamic geometry software, and computer algebra systems. So why now, over two decades later, is programming returning as an important concern?

Ironically, technological evolution has been cited as a catalyst to both the fall and revival of programming in mathematics education. While the initial outbreak of tools made programming skills seem obsolete, such skills have recently been recognized as essential for understanding how the growing mass of "black boxes" work (Lagrange and Rogalski 2015). The increased importance of tool development in various areas of society is also pushing nations to focus more on training a next generation of technologically-skilled workers (Francis and Davis forthcoming). On these bases, some researchers are continuing in the footsteps of the pioneers cited above (cf. ibid.; Misfeldt and Ejsing-Dunn 2015). Others, recognizing that professional mathematical activity has also been influenced by technological developments, study another potential: programming as a way of engaging students in the computational practices of STEM professionals (cf. Weintrop et al. 2016). Indeed, programming was found to be the second most-used technology by the 302 mathematicians in Buteau et al.'s (2014) survey; and, a 2012 report by the Society for Industrial and Applied Mathematics indicates that "programming and computer skills continue to be the most important technical skill that new hirees [in industry mathematics] bring to their jobs" (p. 25). Programming is hence being revived not just as a didactical tool for helping students develop mathematical knowledge, but also as a potential part of the knowledge to be developed. Examining how professional mathematicians now use programming in their work could therefore bring forth new epistemological grounds for its integration in mathematics education.

But as technology and mathematics have evolved, so too has the activity of programming itself. Available to today's programmer are various approaches (e.g., procedural versus object-oriented), languages, and libraries of routines, capable of accomplishing a variety of goals. The difficulty we faced in solidifying a definition of "programming" in our study was rooted in our perceived diversity of computer uses in mathematics. Even our research team was diverse, including an applied mathematician who had worked in industrial-level software engineering, a research mathematician who used computer algebra systems and *C* to explore new territories of mathematical knowledge, and a student who had learned and used a programming language in university mathematics courses.

In many papers, "programming" appears without an explicit definition; and, amongst those that define the term, there is no consensus. Weintrop et al. (2016), who place programming alongside other practices like "designing computational models" and "troubleshooting and debugging", seem to equate the term with understanding, modifying, and writing code (whether it is ten lines of *Python* or millions of lines of *C++*). Buteau and Muller (2010), on the other hand, define a "programming cycle" involving the design, implementation, testing, and revising of a computer tool; and in their Canada-wide survey, Buteau et al. (2014) labelled "programming" with languages like *Fortran, Java,* and *C++*, placing software like *Maple, R,* and *MATLAB* in different categories. In contrast, Lagrange and Rogalski (2015) couple "programming" with the study of algorithms and consider the activity within the scope of software development.

In the end, rather than fixing our own definition, we decided to inquire about the perspectives held by professional mathematicians. This, after all, could provide insight into the place of programming not only in professional mathematics practices, but also in mathematics education.

The Gap between Professional Mathematics and Mathematics Education

The experiences of students and STEM professionals have been compared extensively over the years; for example, in terms of the knowledge at stake (Madsen and Winsløw 2009), the general kinds of behaviour that take place (Cuoco et al. 1996), the influence of institutional constraints (Watson 2008), and certain social and cultural aspects (Burton 2004). The extent and legitimacy of disconnections has been debated (cf. Issue 28, Number 3, of *For the Learning of Mathematics*), not just those within a level (i.e., primary, secondary, university), but also between levels, with universities predicted as exhibiting the closest connection. And yet, Artigue (2016) writes: "There is no doubt that […] undergraduate mathematics education is poorly connected to the mathematics of today in most universities" (p. 22).

In a subsequent paragraph, she highlights a specific gap: while technology has had a major impact on professional mathematical practices, "in many places, undergraduate mathematics education seems still blind to this evolution, even when those in charge make extensive use of technology in their professional activity" (ibid, p. 22–23). This comment seems to contradict two large-scale surveys. Lavicza (2010), driven by a lack of data on technology integration in universities, collected survey responses from 1103 mathematicians in the US, the UK, and Hungary, which led him to conclude: "the extent of technology use at universities is substantial and […] mathematicians have developed an extensive array of technology-assisted teaching materials and pedagogical approaches" (p.

110). He also found mathematicians' use of technology in research to be the most influential factor on technology use in teaching. Another survey he conducted with colleagues (Buteau et al. 2014) reported similar conclusions in the Canadian context.

Upon closer consideration, however, Artigue's comment and the international surveys may not be at odds. First, not all technologies were found to have almost equal use in research and teaching: the number of Canadian mathematicians who reported using programming in their teaching was less than half those who indicated using it in their research (Buteau et al. 2014). Secondly, while the surveys required mathematicians to indicate the ways in which they use technologies in their teaching (e.g., to visualize concepts in lectures, engage students in experimentation, or develop course materials), comparable data was not collected for research. The studies, with their focus on computer algebra systems and primarily quantitative data, also were not aimed at answering certain programming-specific questions: To what extent are students being asked to engage with programming? How do mathematicians describe and explain this level of engagement? Why might the gap exist?

Buteau et al. (2014) hypothesize that the learning curve for programming is steeper than for other technologies (e.g., computer algebra systems, which contain pre-programmed mathematical tools). Its toll on students' and teachers' time and effort could hence make it a less attractive option in regular mathematics courses. Other obstacles, not specific to programming, have also been conjectured: for example, the time required to create new learning activities, logistical obstacles of curriculum-wide integration, and departmental resources and support. Artigue (2016), reflecting on her own difficulty in maintaining an innovative course, describes the ATD as powerful theoretical equipment for addressing such issues of ecology.

ATD, a Frame for Capturing, Comparing, and Clarifying Practices

Chevallard's (1998) *Anthropological Theory of the Didactic* (ATD) provides a generic framework for capturing the different elements of any mathematical activity occurring in any context. The theory highlights the know-how (i.e., the *praxis*) and the discourse that explains it (i.e., the *logos*), defining an individuals' activity in terms of "*praxeologies*" consisting of four components: *tasks* (e.g., solve a system of linear equations), *techniques* for achieving them (e.g., Gaussian Elimination), *technologies* that justify the techniques (e.g., because row operations preserve the solution set of the system) and underlying *theories* (e.g., Linear Algebra). Following the example of Artigue (2002), we prefer to label any logos (i.e., technology or theory) as "*justification*" to avoid the ambiguity of the word "technology", which often appears as a synonym for computerized tools. We also use her distinction between *pragmatic* justifications (e.g., "I used the technique because it's faster"), and *epistemic* justifications (e.g., "I used the technique because it helps me better understand the concept"). Of particular interest to us were praxeologies that, in some way or another, involve programming, as the associated *tasks, techniques,* and *justifications* could provide some answers about the *when, how,* and *why* of programming use by mathematicians and their students. But this would not necessarily reveal the whole story.

Indeed, an individual's praxeologies do not occur in isolation; they develop under the influence of *social institutions* that normalize certain ways of being, thinking, and doing through various conditions and constraints. From the ATD perspective, the word "institution" is to be interpreted in a wide sense: for example, each mathematics department, composed of a unique collection of mathematicians, will foster certain norms (explicit or implicit), as will any subfield of mathematics (e.g., analysis, geometry, or modelling). These norms are not necessarily fixed; on the contrary, the increased acceptance of new praxeologies can cause a re-evaluation of what is deemed possible or important. We could hence expect the place of programming in a mathematician's or a student's praxeologies not only to be influenced by the place of programming in the institutions where their research or learning take place, but also to influence it, even if these mutual influences might not be of the same strength or apply to the same time scale.

To describe the variable status of praxeologies within institutions, we ended up adding to the ATD framework the categories of practices introduced by Morrissette (2011). During a study of school teachers' methods of formative assessment, the researcher identified three kinds of practices:

1. *Shared Practices*, in which everyone engages because they are intimately tied to the profession and remain unquestioned;
2. *Admitted Practices*, which are not shared by everyone in a profession, but are accepted because they have been shown to be effective by innovative practitioners; and
3. *Contested Practices*, which are not accepted by everyone and are therefore situated at the boundaries of the professional culture.

While these notions were not developed within or for the specific context of mathematics education, categorizing programming-related practices as conventional, innovative, or exceptional in both mathematicians' and students' activity allowed us to provide more nuanced descriptions, comparisons, and reflections. After all, if shared practices represent current conventions, their coexistence with admitted or contested practices can suggest opportunities, or even a desire, to re-evaluate what has become conventional. Combining this with the ATD notion of institutional conditions and constraints provided new insight into why gaps like the one found by Buteau et al. (2014) exist; and this analysis also suggested ways of reducing these gaps, should that be deemed the favorable direction to take.

Research Questions and Methodology

We sought to answer the following research questions:

1. How do research mathematicians define computer programming?
2. What is the place of programming in the praxeologies that form the research activity of mathematicians and the learning activity they offer their students? What are the differences and similarities between these praxeologies?
3. What role do institutional contexts play in the choices of mathematicians concerning the use of programming in their research and the integration of programming in their courses?

To answer these questions, we elaborated a study involving individual semi-structured interviews with 14 mathematicians: 3 women and 11 men of various ages, languages

(French or English), and research domains, working within 10 universities in 3 Canadian provinces (British Columbia, Ontario, and Québec). These interviewees were among 17 we invited to participate based not only on our goal of having a culturally, linguistically, and institutionally diverse sample, but also on our interest in interviewing mathematicians who were "using" programming in their research and/or teaching.

Our choice to conduct interviews with mathematicians was driven by the exploratory nature of our study. We sought a rather large sample of professors to cover, as much as we could, the spectrum of possibilities and reasons for (not) integrating programming in mathematics research and education. It is true that when it comes to students' learning, a professor can describe only the "planned" praxeologies, that is, the ones s/he hopes to help develop in students. Even when a professor plans a learning activity that resembles his/her research in the type of practice it aims at developing, this need not imply that the student develops the practice. Including classroom observations or student interviews would have helped to qualify the learning that actually occurred with the planned activities. However, because of the spread of our 14 participants across Canada and the calendar within which we had to work, such data could not be generated within the constraints of the study. In this sense, our study could not compare directly the actual programming experiences of students and those of their professors. Nonetheless, the exploration of planned praxeologies seemed a useful first step, as it could offer professors' perspectives on a priori analyses of activities given to students; and this could pave the way for future research.

While collecting information exclusively from mathematicians, we still wished to make students' potential activity our focal point of comparison with the mathematicians' research activity. Previous studies, such as that carried out by Madsen and Winsløw (2009), have employed the ATD to study the "nexus" between mathematicians' research and teaching activity, in hopes of understanding how the two activities can be mutually beneficial rather than compete for the time and energy of the researcher-professor. Their initial goal was to compare research work with teaching work and it is in contrast with our goal of comparing what mathematicians do and what students are invited to do in mathematics classes. In fact, the shift in perspective from inquiring about how programming is used by a researcher-professor to carry out his/her courses (e.g., to prepare lectures, illustrate concepts, engage students), to exploring the types of programming experiences encountered by students, constituted an important moment in our research project. And this is why we also talk about the programming to be experienced by the students in their *learning* of mathematics, rather than only the programming used by their professors in their *teaching* of that discipline.

Conducting interviews with mathematicians was also the only way to gain access to the justification component of the praxeologies under study. One could question the validity of self-reporting in outlining the praxis components (i.e., the tasks and techniques). To minimize bias, the interviews were based on Vermersch's (2006) *entretien d'explicitation* (*explication interview*, i.e. interview that seeks explicit accounts of experience), in which the interviewer guides the interviewee into a state of descriptive verbalisation where they "relive" specific experiences. The hope is that by becoming re-immersed in those experiences, the interviewee will be able to most accurately describe and reflect on them. Our participants were encouraged to relive moments throughout their research and teaching, in relation to computer programming. Some, in response to our invitation, shared resources they had developed (e.g., computer programs and activity outlines); this enhanced their descriptions.

Asking the mathematicians to describe their experiences led some of them, without direct prompts, to reflect on their choices and, in some cases, contemplate making changes to their practices. Still, some reflections on mathematics, programming, and institutional constraints were solicited to gain insight into the mathematicians' definitions of programming and help us identify factors that could be shaping the praxeologies described. The potential influence professors have on their own teaching and, through the development of curricula, the teaching of others, again supports our decision to collect their perspectives: the issues they consider to be important are likely the ones that should be addressed if an institution wishes to implement any changes and reduce any gaps.

The interviews were recorded, transcribed, and examined through a categorical analysis (Van der Maren 1996, Chapter 19) with a mixed coding approach to identify, classify, and compare the main ideas. Examples of programming use underwent a supplementary characterisation using the ATD framework to extract and compare the types of tasks, techniques, and justifications defining the related research and planned learning praxeologies. Those familiar with the precision of ATD analyses carried out in previous studies (cf. Winsløw et al. 2014) might be surprised by how broad our praxeologies are: they take a sort of bird's eye view on the participants' mathematical activity. This said, it has been questioned if students' and mathematicians' praxeologies could ever be significantly compared with such a fine-grained analysis (Winsløw 2015). Moreover, taking a broad-brush perspective was natural in our case, as we hoped to better understand the place of programming throughout curricula and across institutions, rather than within specific curricular topics like calculus, group theory, or numerical analysis.

Early in our project, we also began to distinguish certain praxeologies and mathematicians as "pure" or "applied". Although this naturally arising distinction adds an interesting dimension to our results, we were hesitant to use it. We even attempted to recognize that the pure-applied dichotomy is truly a continuum by assigning our participants fictive names in a special way: those beginning with an "A" ("P") would represent mathematicians who were most clearly applied (pure), and B's, O's, and N's would identify mathematicians situated somewhere in between. But even now, the categorization of certain participants remains a little blurry, and should not be taken reducing the complexity of their research, teaching, or mathematical identity.

Mathematicians' Perspectives on Programming

What Is Programming?

The mathematicians we interviewed seem to agree that programming includes any activity aimed at constructing a computer tool (a program) by way of three interwoven tasks of varying importance: the development of an algorithm, the coding of the algorithm, and the verification and validation of the program. This said, there is no unanimity about which human interactions with a computer qualify as programming. Our participants mentioned a whole spectrum of computing activities: from constructing geometric figures in *Geometer's Sketchpad*, finding the primitive of an integral with *Mathematica*, or running an open-source *Sage* program to solve large linear systems, all the way to devising from scratch a thousand-line program in *Maple* or *C*. The blurred boundary of programming we encounter in the literature also appears amongst our interviewees; and this reflects the variety, in terms of size, complexity, and permanence, of the programs they develop and use.

When we inquired about the mathematical nature of programming, we were reminded that “doing mathematics is [also] a very poorly defined activity”. In terms of algorithm development, some participants suggested that it was “most mathematical” when it involved complex, deep, or original thinking (as in the search for an efficient algorithm). Others mentioned the need to consider the physical limitations of the computer as distinguishing algorithm development from mathematical thinking. In general, however, algorithm development was deemed comparable to solving a problem or constructing a proof, and was hence perceived as a mathematical activity. In contrast, the mathematicians were reluctant to declare the coding and testing steps as “mathematics” due to their mechanical nature; and only some eventually recognized these steps as demanding the same kind of rigor and proficiency as performing a long by-hand calculation, or checking the logic of a proof. All in all, most participants recognized a mathematical character in some aspects of programming. But the whole task of programming was principally perceived as a technique (or a tool) for accomplishing more important mathematical tasks.

One could wonder if our wide-reaching definition hindered our ability to draw conclusions about the gap that inspired our research. Recall that Buteau et al.’s (2014) conception of “programming” did not include some technologies that we did include in our definition, such as *Maple*, *R*, and *MATLAB*; these were assigned to categories where no significant gaps were found. With our definition, the researchers might not have identified such a large difference between programming use in teaching and research. Irrespective of the size of this gap, however, there are differences between the usages of programming by mathematicians and their students, which led us to conceptualize the activity in terms of levels.

In the praxeologies described by our participants, we identified six levels at which an individual might interact with programming: s/he may

L0: Strictly observe the results of a computer program (under the direction of someone else);
L1: Manipulate the interface of an existing program (in an extracurricular fashion);
L2: Observe (and analyze) the code of a program;
L3: Modify existing code to accomplish something new;
L4: Construct the code of a program, with some elements (e.g., the algorithm) provided; or
L5: Create a program, including algorithm development, coding, and verification/validation.

These “levels” provide a measurement of the visibility of and involvement in the programming activity of a given individual. They should be interpreted as overlapping intervals on a continuous spectrum. For instance, depending on the amount of independence one has in completing the different subtasks of programming, an activity could be classified as L4 or L5; and L5’s most active involvement would include the decision that a task calls for programming, followed by the creation of a program from scratch. Moreover, a single research or learning experience might demand engagement at several levels and include programming activities beyond the span of L0 to L5 (we return to this in the discussion).

Nonetheless, the six levels above enabled us to make significant comparisons between the praxeologies of a mathematician and those intended to be developed by his/her students.

Programming Levels in "Pure" Problem Solving

In both research and learning, we found that a main type of task where programming may be involved is the exploration of abstract concepts, properties, or theories, typically realized through an *Exploration Cycle* including the observation of mathematical objects, as well as the formulation and verification of conjectures. Our study shows that the place of programming within the associated praxeologies may differ significantly between mathematicians and their students.

Like several pure mathematicians interviewed, Omar develops computer tools (L5) to collect evidence about the behaviour of the abstract objects he studies (e.g., in representation theory). He proceeds through an *Exploration Cycle* to first gain the insight necessary to formulate plausible conjectures and then build confidence in their truth. He justifies his technique principally in an *epistemic* fashion, exclaiming: "Before starting to prove something, you'd better know it's true beyond a doubt!" The *pragmatic* character of his programming is also undeniable: he is free to control every aspect of his exploration and extend it to any number of otherwise tedious or impossible examples.

Given his familiarity with creating programs to assist in his own discovery of mathematics, it is not surprising that Omar, like many of our participants, also develops tools to support his students' understanding of challenging notions (e.g., spanning sets and linear independence). From the students' perspective, however, the proposed praxeology is quite different from their professor's: the students are invited to observe dynamic images produced by their professor (L0), are prompted to make guesses, and are provided images that verify or refute their voiced conjectures. Their exploration is strongly guided and limited to the time available in class, and the programming activity remains completely inaccessible to them. The fact that Omar does not share his programs with his students parallels the way he (and other pure mathematicians) communicate their research results: once they have arrived at a theorem and its proof, they typically see no need to discuss the programming that assisted their exploratory work. Similarly in teaching, Omar sees no need to encourage further exploration with a program once he believes the main goal of student discovery has been achieved. Indeed, he emphasizes the *epistemic* quality of the proposed technique, claiming that observing carefully chosen computer-generated results enables students to have their intuitions challenged and the abstract theories they're learning rendered more concrete, interesting, and memorable. When he attributes a *pragmatic* value to the technique, it is in relation to himself: the examples he generates would be difficult, if not impossible, to reproduce on a board, and he would not have the same flexibility of re-executing programs in response to the needs of his students. A summary of Omar's research praxeology and the counterpart praxeology he proposes to his students is given in Table 1.

Paul is a probability professor whose table of praxeologies would differ from Omar's in terms of techniques and justifications. In his research projects that require the use of computer tools (not all of them do), he remains at L0 or L1; the programming is done by a collaborator. And yet, he encourages his students to write and use programs (L4 or L5) to discover content of his probability course. Like Omar, Paul brings forth principally *epistemic* justifications for his students' programming-related praxeologies; the difference is his claim that

Table 1 Omar's research praxeology vs. the praxeology proposed to his students

	Task type: Discover mathematical concepts, properties, or theories	
	Technique	Justification
Omar	L5 + *Exploration Cycle* until "sufficient" evidence collected	*Mainly Epistemic*: Gain insight to formulate conjectures and confidence to proceed to proof
Omar's students	L0 + *Guided Exploration Cycle* until time runs out in class	*Mainly Epistemic*: Have intuition challenged and abstract theory rendered concrete, exciting, and memorable

> It's much better if the students can program it for themselves. If they're sitting in front of the screen and they can play with it and they can adjust parameters, it becomes a kind of a game and it's more interactive for them. And it's better than me just showing them a picture at the front of the classroom, you know what I mean? If they're doing it themselves, they learn it way better.

We will return to this idea that higher-level interactions may have greater *epistemic* value. For now, this claim naturally raises the question: why doesn't Omar invite his students to reach L5?

Ironically, it is Paul who provides an enlightening story for framing the response to this question. It turns out that the probability course described above is geared towards science students; in the corresponding course for mathematics majors, computer technology is absent. Of course, there are many ways to "discover mathematical concepts, properties, or theories", and Paul's pure mathematics students are encouraged to adopt more "traditional" techniques. Upon reflecting on the addition of programming, Paul concluded: "I think it's the right way to go actually. I think that we're missing an opportunity here." So, why not take the opportunity? At first, Paul discussed curricular constraints: the pure mathematics students may not have the prerequisite knowledge needed for programming and it would take too much time to develop and integrate new activities into the already jam-packed curriculum. Omar also explained that "There's so much material in [his] course that it seems like it would be an exaggeration to ask them to program as well […] But, if we had more time, well it would be nice." Since Paul already overcame curricular obstacles during the transformation of the probability course for science students, it seems like something deeper could be at play. He eventually revealed that "Academia is a very conservative place. And there's a huge amount of inertia. And there's also a huge amount of independence among the different instructors." He added: "I don't hear a lot of people talking about this being a great idea." Omar elaborated on similar institutional constraints: "I realize that my department is very abstract […] And the students in pure math love that. But I believe it limits some of their abilities that are absolutely essential if they want to become researchers."

Programming Levels in "Applied" Problem Solving

When it comes to solving "real-world" problems, the applied mathematicians we interviewed create computer programs to accomplish several tasks, including data analysis (to develop or validate mathematical models), parameter calculation (to specify models to a context), or the exploration of model behaviour (for validation purposes or to describe, explain, or predict real-world phenomena). As above, we discuss praxeologies related to one (the last) type of task.

According to our applied participants, exploring the behaviour of their mathematical models necessitates programming for *pragmatic* reasons: not only does it create tools capable of performing a massive number of calculations and experiments (i.e., varying parameters to consider different scenarios), but first and foremost it permits the simulation of models that lack analytic solutions. As Alice explains, “it's highly unlikely that a mathematical model will give you the quadratic formula in the end. It would be nice, but that doesn't happen. And so, computer programming is essential.” Though it was not emphasized, the *epistemic* character of programming is also clear: the visual and dynamic output generated by the computer enables the recognition of patterns leading to conclusions about the phenomena under study.

Given their *pragmatic* justifications, the applied researchers all interact with programming at the highest level (L5). We also observed the least dramatic differences between programming levels in research and learning within this category. Still, there were some notable differences and interesting debates. Barbara’s students, for example, are not asked to develop their own programs. In addition to observing results shown by their professor in class (L0), they receive explanations of her code (L2), manipulate her programs at the interface level (L1), and modify them (L3) to analyse different models. Barbara explains that “[she] want[s] [students] to see that programming isn't that bad. You can do lots of interesting stuff with just a few lines of code.” Through the proposed techniques, her students may learn about programming (e.g., syntax and structure), and may come to appreciate the computer as a powerful tool. Having some insight into the code may also support their understanding of the corresponding output and models. Nevertheless, Barbara justifies mid-level interactions by saying things like, “It wasn't so much how to program a vector field, it was how to use a vector field to understand the model.” Her ultimate goal is for students to understand models and learn how to use them, not necessarily how to program them. Like Omar’s table of praxeologies, Barbara’s would exhibit a gap: L5 in her research vs. L0-L3 in the praxeologies she proposes to her students.

In comparison, Table 2 below shows no difference between the technique components of Ben’s research praxeology and the praxeology proposed to his students (though the students often receive some direction, at least in the form of knowing that programming is necessary).

Table 2 Ben’s research praxeology vs. the praxeology proposed to his students

	Task type: Understand the behaviour of a mathematical model	
	Technique	Justification
Ben	L5 + *Experimentation* (i.e., variation of parameters to observe different output)	*Mainly Pragmatic*: Otherwise impossible due to a lack of analytic solutions and the number of calculations and scenarios to consider
Ben’s students	L5 + *Experimentation* (i.e., variation of parameters to observe different output)	*Mainly Epistemic:* L5 leads to a deep understanding and control of the tool, output, and model

This similarity in praxeologies can be explained by Ben’s belief that L5 programming interaction has a higher *epistemic* value. He explains that “it’s very hard to write a program and not understand what it’s doing. You know, it’s a different level of comprehension.” In contrast, he describes his students’ manipulation of a pre-developed program (L3) as “an exercise in typing. They really didn’t know what it was doing or why it was doing it.” In Ben’s view, if students write their own programs, it is more likely that they will understand the tool, be able to adapt it to their needs, effectively interpret the results, and, by extension, better understand the models. Other mathematicians add that students may come to grasp better the concepts and processes they must structure into an algorithm and transpose into a programming language. Then, having created their own tool, students may feel a sense of empowerment and excitement that may further enhance their experience. Finally, students may gain insight into elements of programming (e.g., algorithms, data structures, code efficiency) that could not only allow them to make better use of existing software (previously “black boxes”), but also provide them with the knowledge to develop their own tools in the future. After all, the more the power of programming is shifted into the hands of students, the more they may be convinced of both the *epistemic* and *pragmatic* value of such techniques. In sum, many participants agree with Paul that “it's much better if the students can program it for themselves.”

Once again, we may wonder why Barbara does not ask her students to develop their own programs. Throughout her interview, the professor complained that her university lacks a mandatory training in programming for mathematics students and that the activity is not widely implemented by her colleagues; some of her students are even afraid of programming. In contrast, learning and using programming are integrated throughout the curricula in Ben’s department. But, as Ben explains, echoing Paul, this systematic institutionalization of programming is not necessarily easy to achieve:

> There's a lot of inertia in Universities. [...] You don't just introduce something and it happens. [...] You introduce it one year, and everybody talks about it, and it's a no. And then there's lots of conversations about it [...] because you want people to have something that they truly need, and that has to evolve through discussion.

Even when the discussion leads to department-wide recognition of programming, the institutional context may impose other constraints. Alice, for example, works at a university that has integrated programming-based techniques, but that lacks the resources for an adequate grading of students’ code; and in Alice’s view, “if it’s not assessed in detail, the requirement is shallow.” As a result, she provides pre-developed programs and asks students to make modifications (L3), much like Barbara. Reflecting the way that she and other mathematicians we interviewed had to develop their programming competencies independently of their mathematics courses, Alice concluded bluntly: “those [students] who take the extra step will learn. Those that don't won't.”

Discussion

The differences between the praxeologies of individual mathematicians and their students seem to align with differences in the status of programming (contested, admitted, or shared) in the related institutions. For our pure mathematician participants, it appears that programming (L5) is *admitted* within their practices: it is becoming more accepted as mathematicians demonstrate how the computer can be useful in pure problem solving. But there still exist pure mathematicians who use only traditional methods. In addition, whether or not programming is involved from one project to the next, can vary greatly for a given pure mathematician, depending on the nature of the project. Phillippe summarizes the general opinion when he says that "Programming is really one tool among many others to do mathematics [...] that is not necessary, but that is useful." In contrast, Barbara suggests that "It's absolutely indispensable for applied mathematicians. I mean, I don't think there's any applied mathematician who doesn't use the computer." Our applied participants explained that their projects simply wouldn't be possible without the computer. Many of them even seemed surprised by the question of how they use programming in their research. Programming (L5) might hence be categorized as *shared* within the applied mathematics community.

When it comes to pure or applied courses/programs offered by mathematics departments, programming is not always accorded as high a status as in the relevant mathematical community. Both Omar and Paul, for example, suggest that their departments *contest* the integration of programming in courses geared towards pure mathematics students. In fact, all our participants agreed that programming should not be accorded a significant place (if any) in courses like topology or measure theory. In contrast, Paul's university chooses to introduce programming to its science students taking a course that is equivalent to one in the pure mathematics program, which likely reflects the greater importance attributed to programming in the applied mathematics community. Indeed, when we asked which courses would most naturally include programming, those related to applied mathematics seemed to be the "obvious" responses. Examples like Barbara and Alice, however, show how institutional constraints might cause programming to be seen as *admitted* rather than *shared* within courses of an applied nature. In sum, we find that while programming (L5) may be part of the *shared (admitted)* practices of applied (pure) mathematicians, within educational institutions it may be portrayed as *admitted (contested)* for the community of students in applied (pure) mathematics courses.

As alluded to previously, many factors may contribute to this status gap. At a personal level, the professors we interviewed all share the goal of assisting students in acquiring mathematical knowledge; what they debate is the appropriate degree to which students should be asked to interact with programming to reach this goal. While some defend lower-level programming approaches as allowing students to effectively gain access to the mathematics, without focussing too much on the programming, others encourage students to engage in programming because they believe it will better support their understanding of the related mathematics. This said, it is interesting to note that Paul and Ben, who are part of the latter mindset, teach in courses that have been specifically designed to implement programming-related praxeologies; in other words, programming has been institutionally recognized as *part of the mathematics* they teach, rather than just one possible *means to the mathematics*. In comparison, professors who implement lower-level programming interactions, such as Omar and Barbara, speak constantly about the institutional constraints they face, including a lack of time to introduce competencies that do not figure amongst the prerequisites or objectives of their courses.

It would seem, then, that an explicit recognition of programming (e.g., throughout curricular documents) as a worthwhile mathematical skill to be developed by students, could be an important factor influencing the status gap. Note that to "contest" a practice in a course is not necessarily a negative standpoint; it is to be expected that not all courses in a pure or applied mathematics program would be deemed suited to programming integration. But if a university wishes to eliminate or reduce the gap, then it seems essential to have courses where programming activity is not only perceived as an appropriate optional addition, but is also institutionally recognized as part of the course; and this could call for a re-evaluation of the mathematics that is taught. It might also require a shift in how programming is perceived by mathematicians: recall that our participants tend to see programming as a tool used to access the heart of their mathematical work. If they were to see it as more central (e.g., worthy of publication), maybe it would be easier to encourage an analogous change in universities. To be critical, perhaps the first step is to decide if (and where) reducing the gap is the favourable direction to take. From Morrissette's (2011) perspective, the work of innovative practitioners, like some of our participants, could make all the difference. As a case in point, the research of mathematicians like Bailey and Borwein (2005) undoubtedly contributed to computer-based techniques evolving from *contested* to *admitted* in the pure mathematics community.

Turning a critical eye on our own work, we realize that some readers might be led to conclude that students should, whenever possible, engage at the highest programming level, L5 (i.e., creating a computer program from scratch). This is not our intention; on the contrary, each of the "levels" may have its own place in undergraduate mathematics education, as it has in mathematical research. The researchers we interviewed mentioned having to interpret their colleagues' computer-based results (L0), using ready-made software so as not to "reinvent the wheel" (L1), and making sense of existing code (L2), either to validate it (e.g., in the case of a collaboration) and/or with the goal of reworking it to fit their own projects' needs (L3). In a society where programming is often collaborative, the lower to mid-levels (L0-L3) could be equally important. In fact, this leads us to envisage another level: L6, write a program to be shared with others. As some of our participants explained, it is quite a different task to prepare a program to be viewed, critiqued, modified, and used by someone else. But this additional level (i.e., sharing a computer tool) could also contribute to the greater visibility of programming in mathematics, with enhanced means for its validation; and this could favour its recognition as a mathematical activity.

Conclusions

In this paper, we presented some results of an exploratory study that sought to better understand why Canadian mathematicians report using programming in their teaching much less than in their research (Buteau et al. 2014).

Our analysis of interviews with 14 mathematicians shows that the word "programming" has diverse meanings in mathematics. Furthermore, when professors declare "using programming" in their courses, their students may interact with the activity at various levels, from strictly observing the results of programs (L0) to independently developing their own computer tools (L5). The identification of six levels allowed us to note important differences between the praxeologies of individual mathematicians and those they propose to their students, with the latter rarely involving the most active version of L5. Adopting Morrissette's (2011) terminology, we further noted that while programming (L5) is *shared* or *admitted* within applied or pure mathematics research communities, respectively, it may be *admitted* or *contested* within institutions where mathematics is taught and learnt.

Adding their views as to why such gaps occur and, by extension, pointing to issues that should be considered when seeking to harmonize praxeologies, our participants spoke of different kinds of institutional constraints: curricular (objectives, prerequisites, time), departmental (tension between academic freedom and course coordination, lack of resources to assess programming in large classes), and cultural (deep-rooted traditions in mathematics). And yet, they spoke equally of the potential benefits of reducing the gaps. Not only might doing so encourage techniques of high *epistemic* value or make students aware of the *pragmatic* character of programming, but it may be important for *social/cultural* reasons: programming may widen students' vision and appreciation of all mathematical activity, while also encouraging the development of practices that could diversify their options beyond their degree.

Of course, one cannot expect the praxeologies developed by university mathematics students to be faithful replicas of professional mathematics practices. This is partly explained by the diversity of these practices and the associated institutions, even among academic mathematicians. But more fundamentally, the learning goals which define the undergraduate experience are not strictly equivalent to research goals. Even with a greater presence of research-like projects involving programming, many learning tasks will still differ from research in scope or approach. Paradoxically, this could mean that for a task where a mathematician engages in programming at a lower level to save time (e.g. by using a preprogrammed tool), a student might be asked to engage at a higher level for epistemic reasons (to understand the associated algorithm, develop skills in programming, etc.).

This reflection emphasizes the need for a nuanced interpretation of our levels. Rather than associating them with a hierarchical scale and systematically aiming for the highest possible, one should choose the level that is adequate for a given task, with a given group of students, in a given course. There could be numerous justifications for exposing students to all levels, including the development of a range of widely applicable competencies: e.g., managing their and others' computational work, determining when a situation calls for programming in the first place, researching and judging the adequacy of existing tools, constructing powerful tools when adequate tools do not already exist, and sharing tools as contributions to society. Given the recent push to implement computational thinking across all school levels, such competencies could be as beneficial for future teachers and professors as for those seeking careers in STEM disciplines. Further research may be needed to uncover what benefits and obstacles these different groups of students *actually* experience while programming in mathematics – but not without continuing to follow the evolution of mathematical praxeologies both within and outside mathematics departments.

Acknowledgements An earlier version of a portion of this paper was presented by Broley at INDRUM 2016 and published in the corresponding proceedings (see Broley 2016). We thank the discussion group on Teachers' Practices and Institutions for their insightful questions and comments throughout the conference, many of which contributed to our expansion of the paper. We also thank the reviewers and editors for their thoughtful suggestions throughout the review process. The research discussed herein was completed as part of a M.Sc. in mathematics (see Broley 2015 for the complete thesis) at Université de Montréal, with the support of the Social Sciences and Humanities Research Council of Canada.

References

Artigue, M. (2002). Learning mathematics in a CAS environment: The genesis of a reflection about instrumentation and the dialectics between technical and conceptual work. *International Journal of Computers for Mathematical Learning, 7*, 245–274.

Artigue, M. (2016). Mathematics education research at university level: Achievements and challenges. In E. Nardi, C. Winsløw, & T. Hausberger (Eds.), *Proceedings of INDRUM 2016 first conference of the international network for the didactic research in university mathematics* (pp. 11–27). Montpellier: University of Montpellier and INDRUM.

Bailey, D. H., & Borwein, J. M. (2005). Experimental mathematics: Examples, methods and implications. *Notices of the AMS, 52*(5), 502–514.

Broley, L. (2015). *La programmation informatique dans la recherche et la formation en mathématiques au niveau universitaire*. Université de Montréal, Montréal: Unpublished master's thesis Available at https://papyrus.bib.umontreal.ca/xmlui/handle/1866/12574.

Broley, L. (2016). The place of computer programming in (undergraduate) mathematical practices. In E. Nardi, C. Winsløw, & T. Hausberger (Eds.), *Proceedings of INDRUM 2016 first conference of the international network for the didactic research in university mathematics* (pp. 360–369). Montpellier: University of Montpellier and INDRUM.

Burton, L. (2004). *Mathematicians as enquirers: Learning about learning mathematics*. Norwell: Kluwer Academic Publishers.

Buteau, C., & Muller, E. (2010). Student development process of designing and implementing exploratory and learning objects. In V. Durand-Guerrier, S. Soury-Lavergne, & F. Arzarello (Eds.), *Proceedings of the sixth congress of the European Society for Research in mathematics education* (pp. 1111–1120). Lyon: Institut national de recherche pédagogique.

Buteau, C., Jarvis, D., & Lavicza, Z. (2014). On the integration of computer algebra systems (CAS) by Canadian mathematicians: Results of a national survey. *Canadian Journal of Science, Mathematics, & Technology. Education, 14*(1), 1–23.

Chevallard, Y. (1998). *Analyse des pratiques enseignantes et didactique des mathématiques : L'approche anthropologique*. Retrieved from http://yves.chevallard.free.fr/spip/spip/IMG/pdf/Analyse_des_pratiques_enseignantes.pdf.

Cuoco, A., Goldenberg, E. P., & Mark, J. (1996). Habits of mind: An organizing principle for mathematics curricula. *Journal of Mathematical Behavior, 15*, 375–402.

Francis, K. & Davis, B. (forthcoming). Number, arithmetic, multiplicative thinking and coding. In *Proceedings of the Tenth Congress of the European Society for Research in Mathematics Education*. Dublin.

Knuth, D. (1985). Algorithmic thinking and mathematical thinking. *The American Mathematical Monthly, 92*(3), 170–181.

Lagrange, J.-B., & Rogalski, J. (2015). Savoirs, concepts et situations dans les premiers apprentissages en programmation et en algorithmique. In A.-C. Mathé & É. Mounier (Eds.), *Actes du séminaire national de didactique des mathématiques* (pp. 155–176). Paris: IREM Paris.

Lavicza, Z. (2010). Integrating technology into mathematics teaching at the university level. *ZDM. The International Journal on Mathematics Education, 42*(1), 105–119.

Leron, U., & Dubinsky, E. (1995). An abstract algebra story. *The American Mathematical Monthly, 102*(3), 227–242.

Madsen, L. M., & Winsløw, C. (2009). Relations between teaching and research in physical geography and mathematics at research-intensive universities. *International Journal of Science and Mathematics Education, 7*, 741–763.

Misfeldt, M. & Ejsing-Dunn, S. (2015). Learning mathematics through programming: An instrumental approach to potentials and pitfalls. In K. Krainer & N. Vondrová (Eds.), *Proceedings of the Ninth Congress of the European Society for Research in Mathematics Education* (pp. 2524–2530). Charles University in Prague, Faculty of Education, and ERME: Prague.

Morrissette, J. (2011). Vers un cadre d'analyse interactionniste des pratiques professionnelles. *Recherches qualitatives, 30*(1), 10–32.

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

Society for Industrial and Applied Mathematics. (2012). *Mathematics in industry.* Retrieved from http://www.siam.org/reports/mii/2012/report.pdf.

Van der Maren, J.-M. (1996). Le codage et le traitement des données. In *Méthodes de recherche pour l'éducation* (2nd ed., pp. 427–457). Montréal/Bruxelles: PUM et de Boeck.

Vermersch, P. (2006). *L'entretien d'explicitation* (5th ed.). Paris: ESF éditeur.

Watson, A. (2008). School mathematics as a special kind of mathematics. *For the Learning of Mathematics, 28*(3), 3–7.

Weintrop, D., Beheshti, E., Horn, M., Orton, K., Jona, K., Trouille, L., & Wilensky, U. (2016). Defining computational thinking for mathematics and science classrooms. *Journal of Science Education and Technology, 25*, 127–147.

Winsløw, C. (2015). Mathematics at university: The anthropological approach. In S. J. Cho (Ed.), *Selected regular lectures from the 12th international congress on mathematical education* (pp. 859–875). Cham: Springer International Publishing.

Winsløw, C., Barquero, B., De Vleeschouwer, M., & Hardy, N. (2014). An institutional approach to university mathematics education: From dual vector spaces to questioning the world. *Research in Mathematics Education, 16*(2), 95–111.