Research This! Questions that Computing Educators Most Want Computing Education Researchers to Answer

Paul Denny
University of Auckland
paul@cs.auckland.ac.nz

Brett A. Becker
University College Dublin
brett.becker@ucd.ie

Michelle Craig
University of Toronto
mcraig@cs.toronto.edu

Greg Wilson
RStudio
gvwilson@third-bit.com

Piotr Banaszkiewicz
Segmars Piotr Banaszkiewicz
piotr@banaszkiewicz.org

ABSTRACT

The goal of many computing education researchers is to improve how computing is taught and learned. To do that, researchers must engage with teachers, coaches, and mentors who design instructional materials and deliver lessons. However researchers may not be investigating problems that are directly of interest or utility to practitioners, and thus may not deliver results that are as impactful as possible in their contexts. To find out what research most interests today's practitioners, we conducted a two-stage survey. The first stage gathered questions that practitioners want researchers to investigate, and the second stage ranked these questions in terms of importance. We found that today's practitioners are more interested in student behavior, student understanding, and pedagogy than in languages and tools, curriculum, and inclusivity, and that there is little overlap between the questions ranked as highly interesting by researchers and those ranked highly by practitioners. Our results indicate that researchers need to better communicate why the questions they are pursuing are important, look for opportunities to collaborate with those who teach but have little direct connection with research, and examine the relevance of their research questions to practitioners.

CCS CONCEPTS

• **Social and professional topics** → **Computer science education**; **Computing education**.

KEYWORDS

Computing Education Research, Educator Survey, Research Topics

ACM Reference Format:
Paul Denny, Brett A. Becker, Michelle Craig, Greg Wilson, and Piotr Banaszkiewicz. 2019. Research This! Questions that Computing Educators Most Want Computing Education Researchers to Answer. In *International Computing Education Research Conference (ICER '19), August 12–14, 2019, Toronto, ON, Canada.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3291279.3339402

ICER '19, August 12–14, 2019, Toronto, ON, Canada

ACM ISBN 978-1-4503-6185-9/19/08…$15.00
https://doi.org/10.1145/3291279.3339402

1 INTRODUCTION

The goal of most Computing Education Research (CER) is to improve how computing is taught and learned. To accomplish that, insight is not enough: researchers must communicate their discoveries to curriculum developers, teachers, mentors, and other practitioners, and those practitioners must find the discoveries interesting and useful enough to act on them. This does not mean that the interests of practitioners should determine the direction of research, but it does mean that researchers should be aware of who is, and who is not, interested in their work, and why.

To address this question, we conducted a study inspired by research in software engineering [2] to investigate the questions that computing education *practitioners* would most like computing education *researchers* to pursue. The theoretical foundation for this work is Herbart's didactic triangle [13]. Its original form relates teachers, learners, and content; we add a complementary triangle to it relating teachers, researchers, and pedagogical content knowledge (PCK) as shown in Figure 1. The concerns teachers have about choosing appropriate content and motivating learners apply just as well when we consider researchers as the "teachers" and teachers as the "learners". Few would argue that what we teach should be driven entirely by what learners think they want to know, but equally, few would argue that teachers should ignore (or be ignorant of) what learners want. Similarly, the agendas of educational researchers should not be set solely by the needs of practitioners, but these needs should be understood and considered.

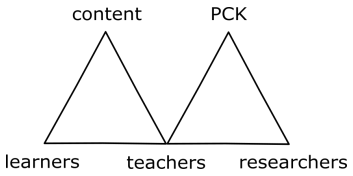

Figure 1: Modified didactic triangle

All data collected in this research are available in anonymized form in the GitHub repository associated with this paper[1] along with the R Markdown document used for analysis; both may be re-used under the Creative Commons – Attribution License.

[1] https://github.com/gvwilson/research-this-data

2 RELATED WORK

In 2012, Daniels and Pears stated: "Computing education research addresses concrete teaching and learning challenges in the discipline drawing on those methods appropriate to the context of the question being investigated" [7, p.101]. CER establishes a connection from researchers to students through practitioners, who are normally teachers situated *between* researchers and students. This relationship was summarized by Malmi et al. in 2014, writing: "CER seeks to gain deep understanding of multiple aspects of the teaching and learning processes of various topics in the computing curriculum, and to build generalizable evidence about problems in students' learning and the efficacy of new teaching approaches to solve these problems." [13, p.27]. They concluded with a list of questions about the role and impact of theories in computing education research, including "How is the acquired theoretical knowledge [developed by researchers] transferred into practical pedagogical content knowledge for computer science teachers?" [13, p.33].

Given the rapid growth of CER literature over the last 50 years [1, 12] and that the results produced by educational research are often unfamiliar to computing educators and difficult to use as a basis for course development [7], there is a need for closer partnerships between researchers and teachers. How researchers develop their research agendas is an issue of interest for the CER community: for example, in a panel session at the 2018 ICER conference, Lewis discussed "the important question of how we pick our research questions, both as individuals and as a community, and how we need broader forums for making these decisions" [10, np].

One approach for identifying research questions of potential interest is to locate gaps in the literature. Kinnunen et al. presented a classification of CER literature using eight main categories derived from a layered model that relates courses, organisations and society [9]. Applying this classification to five years of publications from the ICER conference they found some categories were not well studied, and argue that this approach can help researchers identify areas on which to focus future efforts.

Two recent large-scale reviews of the research literature on introductory programming education revealed the changing trends in topics studied and highlighted areas of interest for future research [1, 12]. These reviews explicitly suggest that their results "may provide insights for future research on how we teach novices and how they learn to program" [1, p.6] and "indicate possible directions for future research" [12, p.55]. Becker and Quille reviewed 481 papers on introductory programming published in the first 50 years of the SIGCSE symposium [1]. Classifying the main topics of the papers published in each decade into eight top-level categories, they observed a sharp increase in recent years in research on students and approaches to learning and assessment, and a relative decrease in research focusing on languages and paradigms.

Luxton-Reilly et al. conducted a systematic review of the research literature on introductory programming published between 2003 and 2017, reviewing 1,666 papers out of more than 5,000 matching their search criteria [12]. They found an increasing number of papers published annually over this period. The authors make a number of suggestions for improving the quality of future research, including reporting more detailed contextual information and strengthening methodological rigor. They also highlight some specific areas where further research is needed, such as identifying particular concepts that might confuse students early in a course, gathering empirical evidence on educational games for programming, and comparing the effectiveness of different interventions for improving student self-belief. An earlier review of novice programming research by Robins et al. in 2003 [16] concluded with its own list of unanswered research questions, mostly centered around programming strategies.

The trends and gaps uncovered by literature reviews provide one mechanism for researchers to identify impactful research questions. In this work, we provide another: surveying computing educators to identify the research questions they most want answered. Surveys are a commonly used instrument in computing education research [12]. One of the challenges with conducting surveys is identifying and sampling appropriately from the target group, which can be difficult when responses are sought from a broad range of participants [3]. Schulte and Bennedsen explored common curricula for introductory programming courses with the goal of creating "a worldwide picture of teachers' opinions about what should be taught" [17, p.17]. Using a web-based questionnaire, they recruited participants through a broad mail-out to recent attendees of computing education conferences such as SIGCSE and ITiCSE, as well as to smaller groups of local workshop participants. They report on the most commonly taught computing topics as well as those perceived by educators as being the most important, but acknowledge the limitations of generalizing from their results.

Elarde and Fatt-Fei used an online survey to explore the content taught in introductory computing courses by members of the SIGCSE and SIGITE communities [8]. They recruited participants through an email to the mailing lists for these communities and report their results under the assumption that "the responses are representative of SIGITE and SIGCSE members" [8, p.58]. Dale explored the topics perceived as most difficult by educators through an online survey sent to 3,500 CS1 instructors [5]. These invitations were sent via the SIGCSE mailing list and through a textbook publisher's database and resulted in around 350 responses. A similar survey on topic difficulty was conducted by Milne and Rowe who contacted educators through a UK-based mailing list for university-level teachers [14]. Despite the biases inherent in collecting data through invitations sent via large mailing lists, such surveys are easy to distribute and permit the gathering of data and viewpoints from a variety of populations [17].

Surveys such as ours are not limited to Computing Education Research. Dancy and Henderson conducted a survey of 722 physics faculty in the United States regarding their instructional practices [6]. They found that despite the fact that most faculty reported knowing about physics education research on curricula and pedagogies and were interested in using them in their teaching, self-reports of actual classroom practices indicate that the availability of these materials has not led to fundamental changes in instruction. They found that faculty reported a shortage of time as being the biggest impediment to implementing more research-based reforms.

In response to the growth in software engineering research, Lo, Nagappan and Zimmerman asked: "how relevant is software engineering research to practitioners in the field?" [11, p.415]. They asked 3,000 industry practitioners to rate the relevance of ideas from brief summaries of recent research papers.

3 METHODS

Our work draws inspiration from Begel and Zimmermann's 2014 study, which generated a catalog of 145 research questions of interest to software engineering practitioners using a two-stage survey process [2]. In the first stage, potential research questions were proposed by one group of participants; in the second stage, another group rated those questions for importance. They deployed the survey to software engineers at Microsoft, sending invitations to 1,500 employees in the question-collection stage and 2,500 in the question-rating stage. They found that questions focusing on how customers typically use applications were of the greatest interest to software engineers, and that there was opposition to questions focusing on the assessment of employee performance. In their conclusion, Begel and Zimmerman stated "we need to think more about the consumer of analyses and not just the producers of them" [2, p.21]. Similarly, we argue that we need to think more about the needs of the practitioner and not just the interests of the researcher. As discussed in Section 1, practitioners' desires should not be the sole driver of research agendas, but they should inform them when appropriate.

We attempted to catalog and prioritize the research questions that computing educators would most like answered using a procedure similar to Begel and Zimmermann's. As in their work, our findings are based on data collected from two surveys: one to collect questions and a second to rate them.

3.1 Participant Recruitment

We wished to include teachers engaged in both formal and informal educational settings working with students of all ages. Our call for participants therefore invited input from all who "…teach programming, web design, or anything else that involves writing code of any kind, in classrooms or otherwise, at any level". We posted this call on social media and on the authors' personal websites, and sent it to mailing lists and forums targeting groups with an interest in computing education, including the SIGCSE, CS Education Research, and CORE mailing lists. K-12 channels included the CSTA and NZDTTA forums as well as various mailing lists for high-school bootcamp teachers. Direct mail was sent to members of the SIGCSE International Committee, through departments and personal networks of the authors, and to approximately 330 non-profit or for-profit coding bootcamps found through Course Report[2].

All announcements sent through these channels described the two surveys that were being conducted and that participants would be assigned at random to complete one or the other. Participants were invited to register their interest in this study by providing a contact email address and told that as an incentive for taking part, a $250 voucher was to be awarded at random to one respondent in each survey. The recruitment process began in September 2018, and after approximately eight weeks 357 participants completed the expression of interest and provided a contact address.

Our solicitation deliberately targeted as broad a range of practitioners as possible, but we recognized that respondents would probably be self-selected to be more involved in research (or at least have more interest in it) than many of their peers. As Schulte and Bennedsen observed more than a decade ago, any study of our community faces similar challenges [17, p.18].

3.2 Survey Implementation

To gather responses, we developed a small web application using Django. The source code is freely available under the open source MIT License, and can be found online[3]. The application's database contained tables to represent questions, the categories to which they belong, demographic information about respondents, and their rankings of selected questions. The rankings table included a hash of the respondent's ID so that we could correlate responses from the same person without knowing their identity.

3.3 Question Collection Survey

We invited 171 randomly-selected respondents to take part in the first survey. At the time of selection this represented half of the participant pool, but while the first survey was being conducted we received an additional 15 responses to our announcement, all of whom were assigned to the second survey. The invitation for the first survey was in the form of an email which included a link to an online form where responses were collected. This email asked: *Suppose a researcher was going to explore questions that are important to you about how computing is taught or learned. What questions would you like answered? Why do you want to know? What would you do with the answers?*

The online form provided space for five questions to be entered along with answers to the accompanying prompts. In addition, four sample questions were shown on the form to serve as exemplars. These seed questions focused on the use of block-based languages, the skills that are often missing in graduates, how to predict success in programming courses, and whether the greatest challenges for self-taught learners depend on age, gender, or socioeconomic background. The GitHub repository for this paper[4] includes copies of all communication sent to participants and the contents of the online forms through which our data was collected.

The survey concluded with demographic questions asking participants what type of teaching they were involved with (schools, colleges/universities, workplaces, free-range programs), the typical age range of their learners, what kind of access to technology their learners had, and what level of involvement they personally had with computing education research. Responses to this latter question were collected on a 5-point scale ranging from "no involvement" to "primary occupation".

A total of 400 questions were submitted by 118 participants (69% of those invited). The most common number of questions submitted by each participant was five (30% of respondents), followed by three (25%), with fewer than 10% submitting just one question. Around 40% of these participants reported conducting computing education research occasionally (26%) or as their primary occupation (14%). Only 7% of participants reported no involvement in research, with the remainder either reading research results or attending research presentations sometimes (33%) or regularly (20%), but not being involved in conducting research.

[2] https://www.coursereport.com/

[3] https://github.com/pbanaszkiewicz/questions_ranker

[4] https://github.com/gvwilson/research-this-data

3.4 Question Consolidation

We reviewed the questions submitted in the first survey to remove duplicates and improve the consistency of wording. In a small number of cases we were not able to understand the meaning of a question and discarded it. In the first phase of this process, two of the authors reviewed 50 randomly selected questions and independently generated top-level category names. We then collectively discussed and refined the proposed categories to generate a set of category descriptors. Two other members of our team then classified the same set of 50 questions using those descriptors and reached agreement on the top-level categories for all questions. Finally, four of the authors independently classified all 400 questions using the categories and descriptors that had been defined. Questions were then grouped by consensus categories to simplify the process of identifying similar questions. Where two or more similar or identical questions were found, we merged them to create a single representative question. For example, the following two original questions:

- Should I teach scratch or other drag and drop methods rather than "pure" programming?
- Do graphical languages (scratch/code.org) really provide a better basis for computational understanding or just draw interest from a younger age?

were merged to create one question in the "Languages and Tools" category worded as follows:

- What are the relative merits of using blocks-based tools vs. textual tools for introductory programming classes?

Our consolidation resulted in a total of 284 questions from the original 400. Table 1 lists the categories, descriptors, and final number of questions for each category following consolidation. A full list of original questions and their mapping to consolidated questions can be found in the data accompanying this paper[5].

The sections below expand on our categories and give examples of typical questions within each of them.

Pedagogy (Computing): 41 questions. Around one quarter of all consolidated questions were related to methods and practices of teaching, which we divided into two categories. The first of these, and the largest category of all, contained questions on approaches for teaching specific computing concepts, such as:

- Do data diagrams help students understand language features and code execution?
- Is explicit discussion of abstract concepts such as patterns or debugging helpful when teaching novices to program?

Pedagogy (General): 25 questions. Other questions relating to pedagogy were more general, focusing on teaching strategies and approaches independent of concepts or specific topic areas:

- How can we help students become comfortable with failure in computing courses?
- What practices are most effective when teaching programming in very large classes (hundreds of students)?

[5] `raw-questions.csv` and `raw-to-final-question-ids.csv`

Curriculum: 36 questions. The next largest category of questions related to curriculum issues, from identifying important concepts that tend to be missing from common curricula to understanding the most effective order for presenting specific topics:

- What is the best way to integrate data management practices into a computing curriculum?
- What are the most important things that students finishing college or university computing programs still don't understand?

Pipeline: 32 questions. Educators are also interested in the pathways by which students enter computer science, how students progress and develop through a formal program of study, and where these programs take them:

- How many students are retained in each level of a typical college or university computing program?
- How do the long-term career prospects for participants in coding bootcamps compare to those of students taking part in traditional classroom computing programs?

Languages and Tools: 31 questions. Questions regarding the tools and languages used for teaching computing were common:

- How do teachers and schools select the programming languages they teach?
- How effective are game development tools like Unity for teaching computing vs conventional languages like Python?

Student Understanding: 23 questions. This category focused on how students come to understand computing concepts, including the misunderstandings they have and how these can be overcome:

- What are students' most common misconceptions about how programs operate?
- What are the differences between the way experts program and the way students program?

Inclusivity: 23 questions. Inclusivity was also a fairly common theme, with just under 10% of the consolidated questions focusing on understanding challenges and improving outcomes for students from under-represented groups or with physical disabilities.

- Do female students learn computing more effectively in all-female classes?
- Has the uptake of for-profit STEM-oriented daycare and camps by middle-class families affected access to computing programs for students from less affluent backgrounds?

Assessment and Predictors: 21 Questions. Questions in this category revealed a strong interest in understanding factors that predict success in computing classrooms as well as approaches for assessing students and measuring learning in different ways:

- How well does participation in non-programming activities (course attendance, reading, discussion) predict outcomes in programming courses?
- How well do standardized secondary school exams predict how students will do in their first college- or university-level programming course?

Student Behavior: 20 questions. A number of questions related to students behaviors and attitudes, including when and how they work.

Table 1: Categories and descriptors used during question consolidation, and resulting number of distinct questions (n)

Category	n	Descriptor
Pedagogy (Computing)	41	How should we teach a specific aspect of computing?
Curriculum	38	What are we teaching and what should we teach?
Pipeline	32	How do learners come into class and progress through them, and where do they go next?
Languages & Tools	31	What software do we use and why?
Pedagogy (General)	25	How should we teach in general?
Student Understanding	23	What higher-level understanding do learners construct?
Inclusivity	23	What differential forces are at work/how do we deal with them?
Assessment & Predictors	21	How do we measure how well we and our learners are doing and how can we predict success in class and after?
Student Behavior	20	What do learners actually do?
Teacher Development	16	How do we train instructors?
Awareness & Broader Impact	16	How do learners find out about us/what do they believe and how does teaching fit into society in general?

- When and why do students ask for help in labs, and are such questions a sign that things are going well or not?
- What benefits do social aspects of classes (e.g., collaboration) have on novice programmers?

Teacher Development: 16 questions. The two smallest categories each contained 16 questions (around 5% of the consolidated questions). The first of these focused on teachers, primarily effective strategies for training them but also exploring how teachers can collaborate with industry partners and computing education researchers:

- What are the most important things for teachers to know about computing education research in order to collaborate effectively with researchers?
- What are effective methods for "training the trainers", i.e., for teaching people how to teach others to teach computing?

Awareness and Broader Impact: 16 questions. The second of the two smallest categories included questions about how students become aware of and perceive computing as a discipline as well as how the teaching of computing impacts on society more broadly.

- Do major news stories about technology and technology companies affect enrolments?
- How important do students who are not already in computing programs believe it is to learn a programming language?

3.5 Question Rating Survey

After populating our survey tool with the consolidated questions, we sent out invitations to take part in the rating process. These were initially sent to the 186 participants who were not selected to take part in the question-collection survey. We later relaxed this restriction and extended the invitation to this final stage of the research to all recruited participants.

Each participant was shown a total of 40 questions selected at random from the set of 284 questions. These questions were presented on two separate pages of the online survey in sets of 20. Participants were prompted with the following statement:

> For each of the following questions, please indicate how important it is in your opinion to have a computing education researcher answer this question.

and asked to rate each question on a six-point scale: *Don't understand*; *Very unimportant*; *Unimportant*; *Indifferent*; *Important*; *Very important*. The question-rating survey concluded with the same demographic questions as the question-collection survey.

4 ANALYSIS AND RESULTS

For analysis, we exported the database tables to CSV files and created an R Markdown notebook to load, clean, analyze, and visualize our data [18]. First, we noted that people either provided *all* of the demographic information or *none* of it. Second, 740 of 7,200 rankings of questions were empty; closer examination revealed that all of these came from people who had not ranked any questions (i.e., had started and then quit) except for one person who had completed one set of rankings (20 questions) but not the second. This person had not filled in any demographic questions, so we discarded their data.

The number of rankings per question ranged from 13 to 33, with a peak at 24 rankings per question. The distribution of the number of rankings of each type by question showed that rankings skewed toward the high end. For example, more than 130 questions had no "very unimportant" responses at all (Figure 2, upper left), while "indifferent", "important", and "very important" had broader profiles.

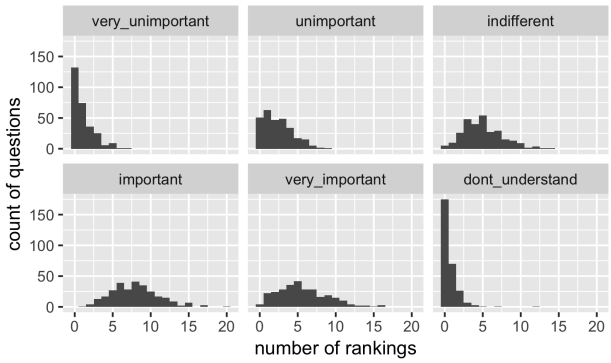

Figure 2: Distribution of scores for each type of ranking

Most respondents gave few or no "don't understand" responses (Figure 2, lower right), indicating that the questions were generally well understood. One notable exception to this was the question: "Is 'expression' a threshold concept?" Half of the 24 respondents didn't understand it; the next nearest fraction for any question was 23%,

and most response fractions were very low or zero. It is possible that "threshold concepts" are not well-known by our respondents. We removed "don't understand" responses from further analysis.

Questions about student access to broadband and/or computers at home or at school showed that almost all respondents work with students who have reliable access at both. A follow-up study should do more to include respondents from less privileged environments.

For the demographic questions, we asked each participant to indicate how many hours they spent on each of the seven types of teaching listed in the decision tree shown in Figure 3.

```
classes = {
    "teaching_children_in_schools" => "young_formal",
    "teaching_teens_in_schools"    => "young_formal",
    "teaching_children_free_range" => "young_free_range",
    "teaching_teens_free_range"    => "young_free_range",
    "teaching_students_college"    => "adult_formal",
    "teaching_adults_workplace"    => "adult_formal",
    "teaching_adults_free_range"   => "adult_free_range"
}
if (person declared one category as "primary activity"):
    class = class associated with that category
else if (person declared both "teaching_children_in_schools"
  and "teaching_teens_in_schools" as "primary activity"):
    class = "young_formal"
else if (person declared more than one category as "primary activity"):
    class = "other"
else if (person declared one category as "hundreds of hours"):
    class = class associated with that category
else if (person declared both "teaching_students_college"
  and "teaching_adults_workplace" as "hundreds of hours"):
    class = "adult_formal"
else:
    class = "other"
```

Figure 3: Decision Tree Used to Classify Respondents

For each category type, respondents selected from *"none", "tens of hours", "hundreds of hours",* or *"primary activity"*. Most reported some kind of teaching as their primary responsibility, while most of the remainder reported that they spent hundreds of hours per year teaching at some level as shown in Table 2.

Table 2: Distribution of answers to time spent teaching (maximum reported time recorded for each respondent)

Teaching time	Number of Responses	Percentage
tens of hours	7	4%
hundreds of hours	39	26%
primary activity	105	70%

For further analysis, we used the decision tree in Figure 3 to classify phase 2 respondents. Three people responded "primary responsibility" for 3, 5, and 7 categories respectively, indicating that they probably misunderstood the intent of the question; they are classified as "other" in our scheme. This categorized respondents as shown in Table 3. We were disappointed that so few of our respondents (slightly over 5%) primarily teach outside formal classrooms, as we feel this is too few to allow meaningful analysis of this group.

To understand the potential impact of having researchers answer the proposed questions, we examined responses to the prompt *"what would you do with the answers?"*. Specifically, we explored

Table 3: Results of Respondent Categorization

Category	Number	Percentage
adult_formal	86	57%
young_formal	40	26%
other	17	11%
adult_free_range	6	4%
young_free_range	2	1%

if participants would make a direct change to their teaching as a result. Two of the authors independently coded the responses to this prompt (with high agreement; Cohen's $\kappa = 0.84$) using a three-category rubric (no action; action unrelated to teaching; direct change to teaching). This analysis revealed that 78% of participants would make a direct change to their teaching practice if at least one of their proposed questions was answered.

To find out which questions were most and least interesting, and to whom, we made three assumptions:

(1) "Very important" is a stronger statement than "important" (and similarly for "very unimportant" and "unimportant").
(2) Ratings are symmetric: "Very important" is stronger than "important" to the same extent that "very unimportant" is stronger than "unimportant".
(3) "Indifferent" ratings have zero weight in respondents' minds.

Given these assumptions, we can score the overall importance of question q as the weighted average of the evaluative scores (excluding "indifferent" and "don't understand" responses):

$$score_q = \frac{(N_q^{VI} - N_q^{VU}) + \omega(N_q^I - N_q^U)}{N_q}$$

where N_q is the total number of responses for question q, N_q^{VU} is the number of "very unimportant" responses for that question, and so on. The parameter ω is the relative weight of "important" or "unimportant" compared to "very important" or "very unimportant"; when ω is 0, the "very" responses are infinitely weightier; when it is 1, "very" carries no extra weight. Setting ω to 0.5 is equivalent to considering each category as equidistant from its neighbours. This is the same as weighting the categories (from "very unimportant" to "very important") in order as -1, -0.5, 0, 0.5 and 1. As we adjust ω, the score for each question changes. Any question that stays in the top N regardless of the value of ω is interesting under any reasonable assumptions about interpretation of our rating scheme.

We arrived at the list of questions considered most interesting overall by computing all possible question rankings for different values of ω and reporting the questions that always appear in the top 20. To allow relative comparison of these top questions, we also report their numeric scores for $\omega = 0.5$. In doing so, we note that it is "appropriate to summarize the ratings generated from Likert scales using means" [4, p.1151], and if the [Likert] numbers are reasonably distributed we can make inferences about their means and differences [15].

Figure 4 illustrates the distribution of numeric scores for all questions (when $\omega = 0.5$) by plotting the mean score for each question against the proportion of ratings for the question that are either "important" or "very important". This proportion is equivalent to the metric *Worthwhile+* as reported by Begel and Zimmermann [2].

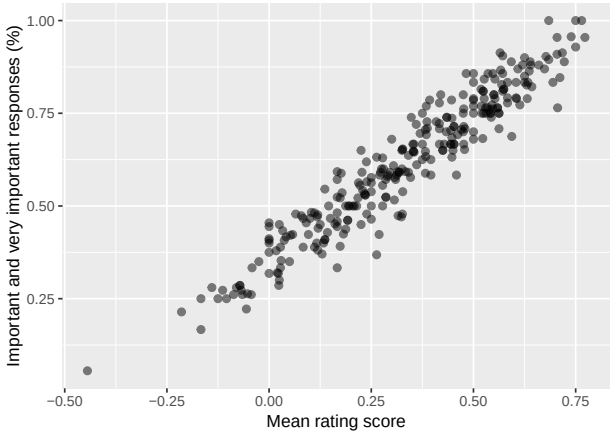

Figure 4: Mean numeric rating score ($\omega = 0.5$) plotted against the proportion of responses for each question that were "important" or "very important"

Most Interesting Questions Overall

The following nine questions are ranked in the top 20 (7%) regardless of weighting. The mean rating score for each question ($\omega = 0.5$) is shown in brackets:

(1) [0.773] What fundamental programming concepts are the most challenging for students?
(2) [0.765] What do people find most difficult when breaking problems down into smaller tasks while programming?
(3) [0.750] What teaching strategies are most effective when dealing with a wide range of prior experience in introductory programming classes?
(4) [0.750] What kinds of programming exercises are most effective when teaching students Computer Science?
(5) [0.739] How and when is it best to give students feedback on their code to improve learning?
(6) [0.717] What affects students' ability to generalize from simple programming examples?
(7) [0.705] What are the relative merits of project-based learning, lecturing, and active learning for students learning computing?
(8) [0.705] What kinds of problems do students in programming classes find most engaging?
(9) [0.677] How can we teach students how to deconstruct programming problems?

These questions belong to just four categories: "Student Behavior", "Student Understanding", "Pedagogy (computing)", and "Pedagogy (general)". To our surprise, no questions from the categories "Languages and Tools", "Curriculum", and "Inclusivity" appear in this list. However, the categories to which the most highly rated questions belong do align with the most common areas in which current computing research, at least on introductory programming, has been conducted. Luxton-Reilly et al. categorized introductory programming research over the last 15 years into several high-level groups [12]. The most common category was research that focused on students, including their behavior and models of understanding, followed by research focusing on teaching and examining the specific techniques and activities used by educators.

Becker and Quille's analysis of 50 years of the SIGCSE Technical Symposium proceedings found that in this decade, the topics Students, Learning & Assessment, and Teaching were the top three topics by a good margin [1]. This study also found that First Languages & Paradigms, Tools, and CS1 Content were amongst the less popular research topics, and is in agreement with our findings of importance here. It appears that the general areas of research that are currently of the greatest interest to educators have been the focus of much of the research to date, at least with respect to introductory programming.

Least Interesting Questions Overall

Using the same ranking method, we found the following 14 questions were the *least* interesting regardless of the relative weight of "very". Again, mean rating scores ($\omega = 0.5$) shown in brackets:

(1) [-0.444] How does the use of Unicode variable names (i.e., mathematical symbols) affect error rates in students' programs?
(2) [-0.214] Is it possible to predict how well someone will do as a computing professional before they enrol in a college or university computing program?
(3) [-0.167] To what extent do people continue to program after moving into management positions?
(4) [-0.167] What technical difficulties do teachers and schools face when trying to get IDEs and other tools installed for students?
(5) [-0.140] Do major news stories about technology and technology companies affect enrolments?
(6) [-0.125] What are the proportions of computing students who identify as male vs those who identify as female?
(7) [-0.114] Are different regions taking different approaches to computing education?
(8) [-0.087] What are the relative benefits of giving students standard hardware vs having them bring their own devices?
(9) [-0.080] What can teachers do to ensure adequate funding for computing classes?
(10) [-0.071] What is the best hardware (such as programmable boards and robots) to use for introductory computing classes?
(11) [-0.071] What impact does early exposure to security topics and tools have on student retention and/or comprehension in Computer Science programs?
(12) [-0.068] What are the relative benefits of provisioning a computing environment for students vs requiring them to manage installation issues themselves?
(13) [-0.065] Should databases be taught at the high school level?
(14) [-0.053] To what degree do different fields of graduate student require or rely on computing skills?

These questions belong to the categories "Assessment and Predictors", "Awareness and Broader Impact", "Curriculum", "Inclusivity", "Languages and Tools", and "Pipeline". No questions in the categories "Pedagogy (computing)", "Pedagogy (general)", "Student Understanding", and "Student Behavior" placed at the bottom of our list, and it is noteworthy that *all* of the top nine questions were in these four categories.

Researchers and Non-Researcher Differences

We then divided respondents who gave demographic information into two groups: those who conducted computing education research either as a "primary" or "occasional" activity (70 people) and those who did not (the other 81). Using the same ranking method as before (questions in top 20 for all ω) we find people who engage in computing education research were consistently interested in:

(1) What fundamental programming concepts are the most challenging for students?
(2) What teaching strategies are most effective when dealing with a wide range of prior experience in introductory programming classes?
(3) What affects students' ability to generalize from simple programming examples?
(4) What teaching practices are most effective for teaching computing to children?
(5) What kinds of problems do students in programming classes find most engaging?
(6) What are the most effective ways to teach programming to various groups?
(7) What are the most effective ways to scale computing education to reach the general student population?

People *not* involved in conducting computing education research were consistently interested in:

(1) How and when is it best to give students feedback on their code to improve learning?
(2) What kinds of programming exercises are most effective when teaching students Computer Science?
(3) What are the relative merits of project-based learning, lecturing, and active learning for students learning computing?
(4) What is the most effective way to provide feedback to students in programming classes?
(5) What do people find most difficult when breaking problems down into smaller tasks while programming?
(6) What are the key concepts that students need to understand in introductory computing classes?
(7) What are the most effective ways to develop computing competency among students in non-computing disciplines?
(8) What is the best order in which to teach basic computing concepts and skills?

Again, these two lists have no questions in common. While there was no overlap in the *specific* questions that researchers and non-researchers ranked highly, there were common *areas* of interest. At a high level, both groups were interested in identifying effective teaching strategies and resources (such as programming exercises) and understanding how to best teach programming to groups outside of the CS major cohort. Both groups were also interested in identifying key concepts: either those that are difficult for students to learn or important for success at the introductory level.

Across all participants, we found that the questions of greatest interest focused on pedagogy and student behavior rather than on languages and tools, curriculum or inclusivity. One possible explanation for this is that people tend to be most interested in things where they can have a direct impact, such as how they personally conduct their own teaching and what they have their students do. Some teachers may have no control over their curriculum (e.g., if it is prescribed at a state level) and so may be less interested in curriculum-related questions. Likewise, creating an inclusive learning environment for their students may be an important personal goal for many teachers. However, if teachers lack direct control over how students are assigned to their classrooms, this may explain their lower interest in research questions focusing on diversity.

5 THREATS TO VALIDITY

We note several threats to validity, including some that emerged as our research unfolded. First, we recruited participants through mailing lists, forums, and a range of private channels in an attempt to target a broad range of practitioners. We did not expect that our sample would be representative of all computing educators across all environments in which computing is taught, but we were disappointed to receive so few responses from people who teach outside formal classrooms. A follow-up study should make a concerted effort to reach more people in this category. Similarly, only a few people reported that their learners do not have regular access to modern computing infrastructure. Any follow-up study should therefore make a greater effort to reach respondents who work with less privileged learners.

To increase the number of ratings assigned to each question, we invited people who had proposed questions in the first phase of this research to also take part in the question-rating phase. As a result, it is possible that some participants were asked to rate questions that they had proposed, which could potentially bias our results. However, given the large number of questions in total, the fact that many questions were reworded or merged, and the fact that no individual participant was asked to rate more than 40 questions, we do not believe this had a material impact on our results.

Finally, we did not anticipate that most people would rank most questions as being of interest (see the distributions in Figure 2). A follow-up study should re-design the ranking process to give a smoother distribution of overall scores. For example, respondents could have been asked to rank questions relative to one another in small batches, and random selection without replacement could have been used to ensure that the number of responses per question was more uniform.

6 CONCLUSIONS AND FUTURE WORK

Our results have given us a great deal to think about. First, they indicate that those who teach programming are more interested in pedagogy and in student behavior and understanding than they are in languages and tools, the specifics of curriculum, or (to our dismay) inclusivity.

Second, there seems to be little overlap between the questions that people who conduct computing education research think are important and the questions considered important by people who teach but do not conduct research. This does not necessarily mean that researchers should change direction, but does suggest that they should examine their assumptions more closely, do a better job of communicating why the questions they pursue are important and foster more communication with those that teach and have little direct connection with research.

REFERENCES

[1] Brett A. Becker and Keith Quille. 2019. 50 Years of CS1 at SIGCSE: A Review of the Evolution of Introductory Programming Education Research. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education (SIGCSE '19)*. ACM, New York, NY, USA, 338–344. https://doi.org/10.1145/3287324.3287432

[2] Andrew Begel and Thomas Zimmermann. 2014. Analyze This! 145 Questions for Data Scientists in Software Engineering. In *Proceedings of the 36th International Conference on Software Engineering (ICSE 2014)*. ACM, New York, NY, USA, 12–23. https://doi.org/10.1145/2568225.2568233

[3] Jens Bennedsen and Carsten Schulte. 2007. What Does "Objects-first" Mean?: An International Study of Teachers' Perceptions of Objects-first. In *Proceedings of the Seventh Baltic Sea Conference on Computing Education Research - Volume 88 (Koli Calling '07)*. Australian Computer Society, Inc., Darlinghurst, Australia, Australia, 21–29. http://dl.acm.org/citation.cfm?id=2449323.2449327

[4] James Carifio and Rocco Perla. 2008. Resolving the 50-year Debate Around Using and Misusing Likert Scales. *Medical Education* 42, 12 (2008), 1150–1152.

[5] Nell B. Dale. 2006. Most Difficult Topics in CS1: Results of an Online Survey of Educators. *SIGCSE Bull.* 38, 2 (June 2006), 49–53. https://doi.org/10.1145/1138403.1138432

[6] Melissa Dancy and Charles Henderson. 2010. Pedagogical Practices and Instructional Change of Physics Faculty. *American Journal of Physics* 78, 10 (2010), 1056–1063.

[7] Mats Daniels and Arnold Pears. 2012. Models and Methods for Computing Education Research. In *Proceedings of the Fourteenth Australasian Computing Education Conference (ACE '12)*. Australian Computer Society, Inc., Darlinghurst, Australia, Australia, 95–102. http://dl.acm.org/citation.cfm?id=2483716.2483728

[8] Joseph V. Elarde and Fatt-Fei Chong. 2011. Introductory Computing Course Content: Educator and Student Perspectives. In *Proceedings of the 2011 Conference on Information Technology Education (SIGITE '11)*. ACM, New York, NY, USA, 55–60. https://doi.org/10.1145/2047594.2047610

[9] Päivi Kinnunen, Veijo Meisalo, and Lauri Malmi. 2010. Have We Missed Something?: Identifying Missing Types of Research in Computing Education. In *Proceedings of the Sixth International Workshop on Computing Education Research (ICER '10)*. ACM, New York, NY, USA, 13–22. https://doi.org/10.1145/1839594.1839598

[10] Andy Ko. 2018. The 2018 ACM International Computing Education Research Conference - Espoo, Finland. https://medium.com/bits-and-behavior/the-2018-acm-international-computing-education-research-conference-espoo-finland-ed548572cca5. (2018).

[11] David Lo, Nachiappan Nagappan, and Thomas Zimmermann. 2015. How Practitioners Perceive the Relevance of Software Engineering Research. In *Proceedings of the 10th Joint Meeting on Foundations of Software Engineering*. ACM, 415–425.

[12] Andrew Luxton-Reilly, Simon, Ibrahim Albluwi, Brett A. Becker, Michail Giannakos, Amruth N. Kumar, Linda Ott, James Paterson, Michael James Scott, Judy Sheard, and Claudia Szabo. 2018. Introductory Programming: A Systematic Literature Review. In *Proceedings Companion of the 23rd Annual ACM Conference on Innovation and Technology in Computer Science Education (ITiCSE 2018 Companion)*. ACM, New York, NY, USA, 55–106. https://doi.org/10.1145/3293881.3295779

[13] Lauri Malmi, Judy Sheard, Simon, Roman Bednarik, Juha Helminen, Päivi Kinnunen, Ari Korhonen, Niko Myller, Juha Sorva, and Ahmad Taherkhani. 2014. Theoretical Underpinnings of Computing Education Research: What is the Evidence?. In *Proceedings of the Tenth Annual Conference on International Computing Education Research (ICER '14)*. ACM, New York, NY, USA, 27–34. https://doi.org/10.1145/2632320.2632358

[14] Iain Milne and Glenn Rowe. 2002. Difficulties in Learning and Teaching Programming&Mdash;Views of Students and Tutors. *Education and Information Technologies* 7, 1 (March 2002), 55–66. https://doi.org/10.1023/A:1015362608943

[15] Geoff Norman. 2010. Likert Scales, Levels of Measurement and the "Laws" of Statistics. *Advances in Health Sciences Education* 15, 5 (2010), 625–632.

[16] Anthony Robins, Janet Rountree, and Nathan Rountree. 2003. Learning and Teaching Programming: A Review and Discussion. *Computer Science Education* 13, 2 (2003), 137–172. https://doi.org/10.1076/csed.13.2.137.14200

[17] Carsten Schulte and Jens Bennedsen. 2006. What Do Teachers Teach in Introductory Programming?. In *Proceedings of the Second International Workshop on Computing Education Research (ICER '06)*. ACM, New York, NY, USA, 17–28. https://doi.org/10.1145/1151588.1151593

[18] Yihui Xie, J.J. Allaire, and Garrett Grolemund. 2018. *R Markdown: The Definitive Guide*. Chapman & Hall/CRC The R Series.