Creation and Validation of Low-Stakes Rubrics for K-12 Computer Science

Veronica Cateté
North Carolina State University
Raleigh, North Carolina, USA
vmcatete@ncsu.edu

Nicholas Lytle
North Carolina State University
Raleigh, North Carolina, USA
nalytle@ncsu.edu

Tiffany Barnes
North Carolina State University
Raleigh, North Carolina, USA
tmbarnes@ncsu.edu

ABSTRACT

With increased numbers of K-12 computing courses, we also see an increase in teachers new to the subject, making it difficult for them to properly assess student programming assignments. Many of these teachers require project-specific rubrics to help assess student learning. Researchers have attempted to create systematic, validated, and reliable rubrics for these courses with only minor success. In this research, we make an argument for the validity of our low-stakes computing rubrics. In doing so, we establish a validated method for creating a full-suite of project-based rubrics for K-12 computing courses, helping teachers, researchers, and practitioners make much-needed course materials. Evaluating these rubrics, we see grader consistency as well as heatmaps of where teachers are looking for computational thinking concepts in code.

CCS CONCEPTS

• **Social and professional topics** → **Student assessment**; **K-12 education**; *Computational thinking*;

KEYWORDS

Nominal Group Technique; CS Principles; Rubrics; BJC

ACM Reference Format:
Veronica Cateté, Nicholas Lytle, and Tiffany Barnes. 2018. Creation and Validation of Low-Stakes Rubrics for K-12 Computer Science. In *Proceedings of 23rd Annual ACM Conference on Innovation and Technology in Computer Science Education (ITiCSE'18).* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3197091.3197134

1 INTRODUCTION

New K-12 computer science (CS) courses and modules are being developed, often based on existing higher education courses or materials from outreach programs [5]. When transitioning these courses to formal K-12 classrooms, simple formative assessment items like rubrics might not be available. This is unfortunate as a large percentage of the new K-12 computing teachers have limited computer science backgrounds [5, 7]. Research has shown that without substantive knowledge on the subject matter, it is difficult for teachers to give helpful or appropriate feedback [7, 12].

ITiCSE'18, July 2–4, 2018, Larnaca, Cyprus

ACM ISBN 978-1-4503-5707-4/18/07…$15.00
https://doi.org/10.1145/3197091.3197134

In previous research [2, 3, 14], several systematic approaches to creating rubrics have been developed. However, these are not cost-effective for the frequent restructuring of CS activities. In this research we propose using the the Nominal Group Technique (NGT) [6] to create low-stakes formative assessments for K-12 computing courses. We hypothesize that this method will lead to improved practicality in designing rubrics at scale. In this paper, we examine the methodologies for developing rubrics in novice CS courses. We present our method for creating assessments, the results of testing the rubrics with novice computing teachers, and our analysis on the effectiveness and validity of these rubrics for low-stakes assessment.

2 BACKGROUND

Recently, research on systematic rubric development has increased, reflecting the need for further instructional support for inexperienced graders [2, 13, 15]. The most common methods for systematically generating rubrics has been a mix of surveys of the field [3, 13] and various group decision making techniques [2, 14, 15] which require a large amount of effort and administration. We examine these current methods and then introduce an alternative methodology (NGT) for timelier group decision making.

2.1 Rubrics in CS

Both Stegeman and our early research attempts at rubric creation focused on in-depth literature reviews [3, 13]. Stegeman's effort examined and cross-referenced code quality handbooks for software engineering, whereas we investigated literature on auto-graders. While these initial attempts were successful, the research trend evolved to focus more on group collaboration [2, 14, 15].

Stegeman et al. revised their rubric scheme through an iterative design cycle involving instructor feedback and think aloud studies. The goal was to design a rubric that had category descriptions which could clarify the differences between levels. Through this, a student could understand "what good performance on a task is; how their own performance relates to good performance; and what to do" to improve their score [14]. Stegeman et al. suggested construct validity through the literature basis used to initially create the rubric, as well as through the intense instructor involvement recurring each iteration cycle.

Our revisit to rubric creation pivoted from the task-oriented rubric developed using auto-grader literature to learning-oriented rubrics generated through Delphi surveys [2]. During this process expert CS teachers reviewed each of the learning objectives presented in the course framework and selected the best fitting ones to be a part of the rubric. Consensus was achieved through 3 rounds of iterative surveys spanning up to 11 weeks. The final rubrics achieved moderately high reliability among test users.

Yuan et al. had a professor create an 'expert' rubric that would be used by a crowd-sourced program of novice graders. Their evaluations showed that without a rubric, students perceived the feedback from expert graders to be significantly more useful than that of novices. However, there was no significant difference in perceived helpfulness between expert feedback and novice rubric-generated feedback [15]. These results only measured perceived helpfulness by students and did not focus on rubric validity or reliability.

Each of the above methods relies on the presence and assistance of one or more experts in the field and a large amount of time is needed to generate each rubric. These methods limit the required scalability of rubric creation needed to meet increased demand for K-12 computing courses. We believe we can improve on these methods by using the more time efficient, Nominal Group Technique.

2.2 Nominal Group Technique

The Nominal Group Technique (NGT), another form of consensus gathering, is a product of 1960s social psychology and is used as a bridge between researchers and practitioners [6]. It relies on the belief that consensus by experts can generate a strong, effective, empirical generalization. NGT utilizes a co-located group of individuals moderated by an authoritative facilitator in order to seek group consensus on applications such as problem identification, question generation, and development of solutions [8]. As rubrics are being developed post curriculum creation, group consensus is important in determining learning objectives. In this version of NGT we added intermediate steps to share and discuss ideas with a partner before sharing with the full group. After this intermediate discussion, partners would rank and vote on their objectives before bringing the problem to the entire group for consensus approval.

3 METHODS

In the following subsections we detail the course chosen for rubric creation (AP Computer Science Principles), our rubric creation process, and the methods for evaluating the resulting rubrics.

3.1 Corpus: AP Computer Science Principles

High school enrollments in computer science courses parallel the explosive demand for college-level computing courses. In 2017 for example, 44,330 students took the new Advanced Placement (AP) CS Principles (CSP) exam, and 60,519 students took the AP Computer Science A exam, tripled from just 20,120 in 2010. Together, students taking AP computer science exams expanded over fivefold in seven years. Such demand requires substantial teacher recruitment and professional development for both courses. In this case study we focus on the newly offered AP CSP course. To meet demand, both course materials and teacher development had to occur rapidly, and teaching requirement conflicts meant many teachers for AP CSP werenâĂŹt those already teaching AP CSA. This resulted in teachers recruited to teach AP CSP coming from a diverse set of backgrounds with few having prior experience in computing.

As universities run professional development with rapidly updating AP CSP materials, we found a need to quickly create reliable, low-stakes formative assessments (rubrics). After talking with relevant stakeholders (curricula developers and professional development teachers), we decided rubrics should be formated consistently across lab assignments and link to learning objectives as outlined in the course framework. We determined that analytical rubrics, giving finer grained plain-text feedback, would be more formative to the students than a holistic rubric with a single assessment metric.

3.2 Rubric Creation

In order to maintain a rigorous rubric while improving upon implementation costs in prior research [2], we implemented a modified NGT process. One advantage is that localized procedures worked much more efficiently with all participants meeting in person. Unfortunately, gathering a group of K-12 computing teachers during the school year is difficult, given how few computing teachers there are per single district. In this section, we detail a new rubric creation process to fit a more practical development cycle. The steps of the modified NGT process are as follows:

1. Background Training
2. Introduce and Explain Topic
3. Generate Ideas Individually
4. Share Ideas in Pairs
5. Discuss with Partner
6. Vote and Rank
7. Share with Group
8. Discuss as Group

3.2.1 Training 'Almost Experts'. Although expert teachers are ideal candidates for creating rubrics, it is not feasible to get them together for a long enough period to create each assignment specific rubric needed for AP CSP. Even with significant compensation, few active teachers had the time available to participate in our past studies. Therefore, we chose to follow Yuan's example and create 'almost experts'. Yuan defines experts as those having an undergraduate degree and work experience in the field (design) and almost experts as novices using expert materials [15]. Using this definition, the novice graders in our previous work were almost experts as the rubrics being tested were created by experts.

We push the concept of using almost experts further by having them create the initial rubrics. To do this we formed a 4-person group of senior CS undergraduates enrolled in an independent research course. We argue since these students have well surpassed the course content of AP CSP they had relative expert content knowledge, but lacked training in the official course guidelines. Further, this strategy was economically advantageous as undergraduate researchers receive course credit rather than financial compensation for their participation.

To prepare the team, we introduced AP CSP and rubric development during two 90 minute training sessions. Once the undergraduate research team demonstrated understanding of the computing concepts and rubric techniques through verbal questioning and examples, we instructed them in their task of creating new AP CSP rubrics for the curriculum's active lab assignments. The team was instructed to make both task and learning-oriented rubrics, however, this paper only focuses on the latter.

3.2.2 Streamlining the Process. In order to systematically create the new set of learning-oriented rubrics, we implemented a modified NGT. In addition to training qualified participants instead of using preexisting experts, we divided the work load among the team, such that each of the seven course units were examined by at least two people before being sent to the group discussion.

After pairing undergraduates, we distributed the course units and major lab assignments (ones that used more than four blocks of code in the final solution). For each major lab assignment, the

pairs were to individually select each of the learning objectives and EK components they felt best matched the assignment. They were told to focus particularly on objectives dealing with programming, algorithms, and abstraction (the most relevant big ideas for labs).

Once each person had a list of learning objectives, they shared and discussed it with their partner. Any objectives that overlapped were automatically included, and those that did not overlap were discussed until the pair could come to an agreement. After each pair of researchers compiled a list of learning objectives for their particular lab assignment, the team of four met back up to discuss any outstanding learning objectives with input from the other pair.

After each lab had learning objectives, the team was taught to cluster related objectives and set anchor points (benchmarks). For each category of the rubric, the team was instructed to come up with a simple description of one or two words and a set of 2-3 objectives. Afterwards, the team created anchor points that would indicate a completed version of the assignment (4) followed by textual descriptions for the rubric levels of above expectations (3), below expectations (2), and needs improvement (1). A score of zero was reserved for no attempt made. Each pair created corresponding rubrics for the lab assignments they investigated earlier in the cycle.

3.2.3 Quality Assurance. Once teams completed their lab rubrics, an undergraduate researcher unified them, resolving differences in wordings for categories (e.g. 'clean-code' vs. 'cleanliness') and anchor values (e.g. 'some abstract functions' vs. 'a few abstract functions') as well as aligning goals. In order to create rubrics quicker while maintaining a theory-inspired systematized procedure, we augmented NGT as described by Delbecq and McMillan [6, 10] to accommodate a more efficient work flow given the number of rubrics needed. After applying our process, 32 learning-oriented rubrics and 34 task-oriented rubrics were created that corresponded to each of the lab assignments in the seven AP CSP units. All 66 rubrics were created over a span of ten 60-90 minute sessions.

3.3 Rubric Evaluation

To evaluate the rubrics, we wanted to compare educators' ability to identify relevant learning objectives directly in student code as well as evaluate whether or not graders are able to use the rubrics consistently.

3.3.1 Participants. Our priority was to recruit newly active high school computing teachers and soon to be AP CSP teachers. We limited participation in the study to those who had taught less than 2 years of AP CSP and who had not taught other programming courses (e.g. AP CSA). We recruited from the CSTA mailing-list, and allowed computing teachers to share this opportunity with their STEM educator peers. We also invited student participants majoring in Science, Technology or Math Education. Due to the length of the study, teacher participants were compensated with an entry into a raffle for a $500 gift card and student participants were given a $10 to $15 dollar gift card depending on the number of other students they referred in the post survey.

3.3.2 Study Procedures. Participants completed a pre-post survey followed by an intermediate reflection activity on rubric use and an optional introduction to the Snap*!* programming environment. Participants completed a brief online pre-survey collecting demographic information, teaching background, and education level. The pre-survey also included a high-level program comparison activity between three rubric-less sample coding assignments.

Part two had participants reflect on their evaluation strategies for the previous samples through targeted questions (e.g. "What metrics did you use to grade the assignments?, "Did you refer to the learning objectives for the course?" etc.) After this, participants were provided with the list of Essential Knowledge (EK) items relevant to the prior labs. Participants were given the descriptions for three lab programming assignments and tasked to pair the appropriate EK with each lab. Participants were then shown one of our sample rubrics and were tasked with re-evaluating the student code, this time with both categorical rubric grades and by highlighting the code that directly correlates with each given grade. There was a link provided to an optional Snap*!* tutorial, which shows them how to run and examine student projects and described how loops and other functions are implemented in Snap*!*.

After participants gained experience in grading Snap*!* student projects, they completed a final post-survey where they graded three samples for each of three different programming labs. They highlighted in each program the code that directly related to the described learning objective for that category of the respective rubric. Participants were given the original lab description handed out to students, a link to the live sample running in Snap*!*, and code snapshots of the relevant student-created functions. For each lab description in the post-survey, we provided one sample each of high, medium, and lower quality student code for participants to review. Descriptions of the lab assignments are listed in the subsection below. These labs were chosen due to their level of moderate difficulty, although Binary includes recursion, the main task for students is to abstract a new base variable. The C-Curve task is more complex and tests the upper bounds of the CS Principles teachers' understanding of different computational thinking concepts.

3.3.3 Lab Assignment Descriptions. The **Beyond Binary** lab extends the original decimal to binary assignment, tasking students to generalize the pattern for conversion using a 'convert to base' block that takes the base as a second input. Figure 1 below is given to students and demonstrates converting decimal to binary. To use any base, a student need only add another input parameter and replace all the 2's in this code with the new variable.

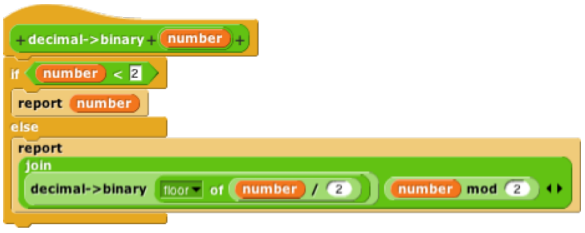

Figure 1: Snap*!* code for Decimal to Binary conversion.

From the **C-Curve** lab description: "We can make very very complex images by just repeating the same shape multiple times. You'll be writing the recursive function to draw the C-Curve.

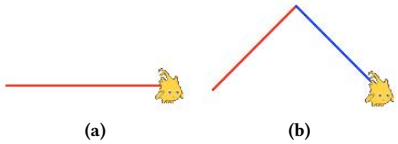

Figure 2: Level 1 and 2 of the C-Curve algorithm

In Figure 2, the base case is a single line. The sprite starts facing right and faces right at the end."

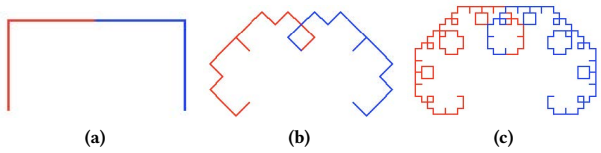

Figure 3: Extended recursion levels of the C-Curve

4 RESULTS & ANALYSIS

A total of 19 participants (9 teachers, 10 students) were recruited. 15 participants (8 teachers, 7 students) completed the Binary Conversion activity, and 14 participants (6 teachers, 8 students) completed the C-Curve lab. Participants were able to stop and continue working at later times . Participants identified as rushing through the material (completion time of less than 2 minutes) were removed. These time filters and some participants failing to complete cause the differing number of participants for each lab. A breakdown of participation demographics is available in Table 1.

Table 1: Participant breakdown for the evaluation study. Participants in the middle row are part of both data sets.

Under Review	Pop	Gender	Occupation
Binary Converter Only	N=5	3 F, 2 M	3 CSP Teachers
			2 Math Ed Majors
Binary & C-Curve	N=10	5 F, 5 M	4 CSP Teachers
			1 Chem Teacher
			3 Tech Ed Majors
			2 Math Ed Majors
C-Curve Only	N=4	3 F, 1 M	1 Business Teacher
			3 Math Ed Majors

4.1 Intra-class Correlations

We investigated reliability statistics between raters. Reliability value ranges between 0 and 1, with values closer to 1 representing stronger reliability. To determine the variance between 2 or more raters who measure the same group of subjects, we use inter-rater reliability. We chose Intra-class correlation coefficient (ICC) to reflect both a degree of correlation and an agreement between measures. Of the 10 forms of ICC, we chose the consistency definition of ICC(2, k) meaning a two-way random model with k raters.

We calculated an overall level of reliability using the 10 core participants across both data samples: binary and C-Curve. ICC estimates and their 95% confidence intervals were calculated based on a mean-rating (k = 10), consistency-agreement, 2-way random-effects model. We calculate our results to be ICC = 0.75 with 95% confidence interval = 0.61-0.86. Based on the ICC results, we concluded that the inter-rater reliability is "good" to "excellent" using Cicchetti's guidelines for reliability interpretations [4].

We next computed reliability for each lab separately. For each of the lab assignments, we calculated the combined reliability of teachers, students, and both participant groups together. Highlights of these results are shown in Table 2 below.

Table 2: Intra-class correlation coefficients (ICC(2,k)) and confidence intervals for Binary and CCurve lab assignments

	Binary Conversion	C-Curve Generation
Teachers + Students	ICC(2, 15) = .90	ICC(2, 14) = .76
	CI[.80, .96]	CI[.57, .90]
Teachers Only	ICC(2, 8) = .87	ICC(2, 6) = .74
	CI[.73, .95]	CI[.50, .89]
Students Only	ICC(2, 7) = .74	*ICC(2, 8) = .39*
	CI[.73, .95]	*CI[-.16, .74]*

4.2 Heatmap Triangulation

As part of the study, participants were asked to identify sections of code that influenced their decision for giving each rating (e.g. highlighting $\sqrt{2}/2$ for evaluating the 'Math' rating). In total, participants made 1000+ indications (540 made by 15 participants for Binary Conversion, 468 made by 13 participants for C-Curve). One participant only completed the grading portion of C-Curve and not the code identification task.

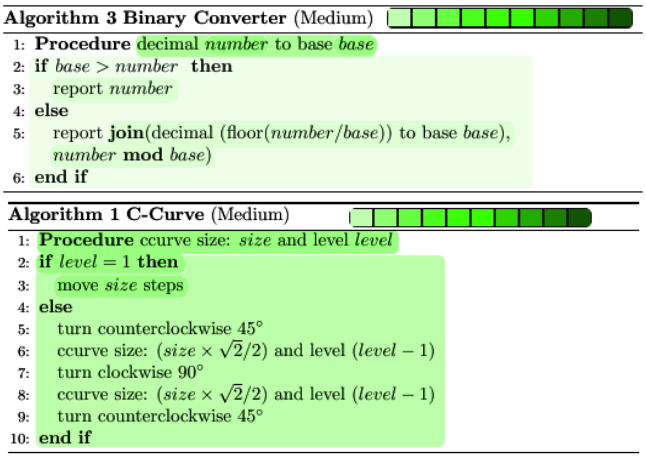

Figure 4: Heatmaps for Name Category in Binary Sample 3 and C-Curve Sample 1

In Figures 4 and 5, we present representative samples of code indication by active teachers. The intensity of the color correlates

to the frequency of its selection. The first code sample shown represents the Snap*!* code for the third Binary Conversion solution participants were given and the second sample represents the first C-Curve Generation solution participants were given.

Algorithm 3 Binary Converter (Medium)

```
1: Procedure decimal number to base base
2: if base > number then
3:    report number
4: else
5:    report join(decimal (floor(number/base)) to base base),
      number mod base)
6: end if
```

Algorithm 1 C-Curve (Medium)

```
1: Procedure ccurve size: size and level level
2: if level = 1 then
3:    move size steps
4: else
5:    turn counterclockwise 45°
6:    ccurve size: (size × √2/2) and level (level − 1)
7:    turn clockwise 90°
8:    ccurve size: (size × √2/2) and level (level − 1)
9:    turn counterclockwise 45°
10: end if
```

Figure 5: 'Math' Heatmaps for Binary Sample 3 and C-Curve Sample 1

4.3 Process Validation

The most prevalent uses of validity involve complex psychometric measurements aimed at standardizing high-stakes testing [11]. In order to evaluate the quality of our low-stake rubric system, we use Baartman's validated framework, the Wheel of Competency Assessment (Figure 6) [1], which focuses on criteria relevant to open-ended project assessment that are meaningful to project stakeholders. Using the criteria outlined in Baartman 2006, we have tested all of the evaluation criteria not directly linked to student assessment.

Center to the framework are the core 5 concepts which must be completed before other criteria can be accounted for. The format of our rubrics complete requirements for comparability, transparency, and fairness. Their basis in the AP CSP curriculum framework give fitness for purpose. We found reproducibility of decisions through our intra-class correlation testing as it pertains to AP CSP teachers. We also found our teacher base as well as our AP CSP professional development partners accepting of the rubrics for assessment needs. Furthermore, our participants expressed increased confidence in being able to provide meaningful feedback to students.

The categories of authenticity and cognitive complexity relate to the assignment rather than the evaluation metric, so they are excluded from this study. The final two metrics, educational consequences and costs are external factors to the assessment. However, using the modified NGT method with almost experts to create the rubrics significantly reduced production cost and increased efficiency when compared to previous development methods for similar rubrics [2, 14].

5 DISCUSSION

This study aimed to test whether we could improve upon methods for developing and validating introductory computing rubrics. To

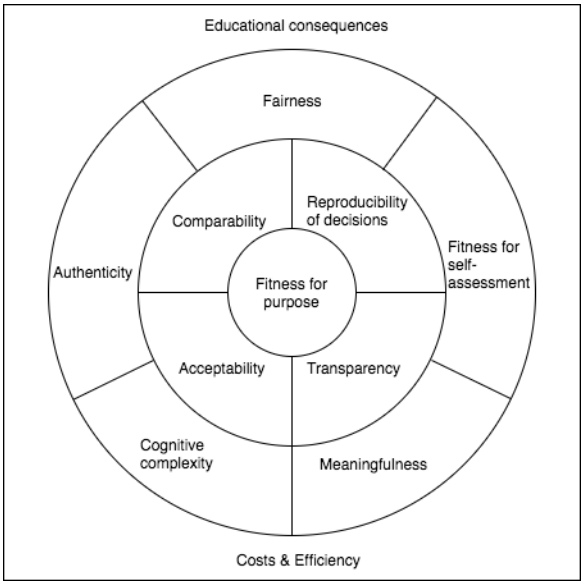

Figure 6: Wheel of Competency Assessment

accomplish this, we trained a new team in the AP CSP curriculum and rubric development, complementing their pre-existing computer science skills. By creating a localized team of almost experts, we were able to adapt the NGT process to be used for rapid rubric creation. We evaluated these new rubrics by testing them with new AP CSP teachers. We discuss the creation and evaluation below.

5.1 Rubric Creation

Our main changes to typical rubric creation protocol were the use of almost experts and modifying NGT to include think-pair-share [9]. NGT relies on qualified participants to make expert judgments and opinions on a given topic. Unfortunately, not all AP CSP teachers have a deep background in computing or the time available to meet for a prolonged period. Most teachers who have participated in AP CSP professional development took only a few computing courses in their undergraduate degrees. We argue that 3rd and 4th year computer science undergraduate students have sufficient computer science expertise to understand simple AP CSP student programs, and that a focused workshop that introduces them to the AP CSP framework is sufficient preparation for rubric development.

We modified the NGT by inserting a shorter refinement step between a pair of participants before introducing ideas to the full group. By submitting full drafts of rubrics rather than individual learning objectives to the group, participants were able to discuss assessment criteria as a whole. This made it so that each lab assignment had a cohesive and comprehensive list of learning objectives that were agreed upon by the team. Additionally, the discussion time spent on an individual rubric was more focused, streamlining the total time spent on development. Overall both of these modifications led to an increase in scalability of rubric creation.

5.2 Rubric Evaluation

5.2.1 ICC. We first investigated inter-rater reliability. Across both projects we achieved an acceptable ICC value of .74. This is

promising as we have mixed levels of coding samples and AP CSP teaching experience (0-2 years). The mixed experience becomes noticeable when broken down by project. For the Binary Conversion lab, each group scored an ICC >0.7, demonstrating that graders are able to score consistently using the rubrics on simple assignments. The C-Curve lab is noticeably harder and experience plays a role in evaluation ability. The ICC for the combined group and teachers is >.70. However, STEM Education students scored an ICC of .38.

Contrary to the high score density in the teacher only data, the student score distribution is polarized. This polarization can be defined by student major; Technology education students gave consistently lower scores compared to Math education students. We believe that as Math education majors tend to take more math classes and have seen recursive functions or fractals before, they were more comfortable grading C-Curve samples. This trend was later identified in the C-Curve teacher data, where the sole Chemistry teacher performed lower than the math oriented teachers.

5.2.2 Heatmap Triangulation. Through the heatmaps, we see how teacher's understanding of rubric categories changed. Teachers tended to include more code elements as being relevant to the rubric category in the C-Curve assignments than Binary (e.g. comparing the selections for "Name" in Figure 4 Binary's heatmap focuses on the procedure definition while in C-Curve, more teachers selected elements of the code that involve the naming of custom functions).

Similarly, in comparing "Math", both heatmaps have high values in traditionally mathematic concepts (e.g. variables manipulation or numerical functions). However, teachers also identified logical control structures (if-else) as "Math" for C-Curve. Furthermore, the presence of proportionally higher selected regions in C-Curve assignments also suggests that teachers agreed more frequently on relevant elements during this later assignment. Although some teachers identified the logical constructs as being part of the math category, which includes mathematical and logical reasoning, the tendency was to select the more explicit math functions.

The overall trend in the heatmap data is that teachers were able to correctly identify the minimum qualifications for each rubric category on correctly functioning student code. Not pictured, are sample heatmaps for low-performing student code. On these samples, teachers selection criteria were more diverse as they were trying to find alternative elements to give partial credit.

5.2.3 Process Validation. Although Messick's psychometric oriented Construct Validity is more widespread, it is primarily used for high-stakes testing and not appropriate for simple and rapidly changing lab assignments needing formative feedback. Instead using WoCA we were able to evaluate criteria important for the use and adoption of our rubrics into new AP CSP classrooms. We found that feedback from teacher use of the rubrics satisfied two relevant subjective aspects of the framework. We also found that our systematic creation of the rubrics and the subsequent reliability testing produced results to satisfy six other framework criteria. The remaining four criteria were not applicable to our scenario, although we believe the nature of the programming assignments lend themselves to being *authentic* assignments utilizing higher *cognitive* skills.

6 CONCLUSIONS

This research suggests that a co-located team of trained CS undergraduates can create learning-oriented rubrics for use by new AP CSP teachers. Our evaluation study shows teachers were able to achieve moderate levels of rater consistency and are on track for identifying computational thinking concepts in code. We found that more support will be needed for those without a strong math background as advanced algorithms use higher levels of reasoning and understanding. We also found that as computing researchers, we have a misunderstanding of what is included in technology education training, and that we need to work more closely with our college of education to specifically prepare new computing teachers.

ACKNOWLEDGMENTS

This material is based upon work supported by the National Science Foundation under grant numbers 1252376 and 1542922.

REFERENCES

[1] Liesbeth KJ Baartman, Theo J Bastiaens, Paul A Kirschner, and Cees PM Van der Vleuten. 2006. The wheel of competency assessment: Presenting quality criteria for competency assessment programs. *Studies in Ed. Eval.* 32, 2 (2006), 153–170.

[2] Veronica Cateté and Tiffany Barnes. 2017. Application of the Delphi Method in Computer Science Principles Rubric Creation. In *Proc. of the 2017 ACM Conf. on Innov. and Tech. in Comp. Sci. Ed. (ITiCSE '17)*. ACM, New York, NY, USA, 164–169.

[3] Veronica Cateté, Erin Snider, and Tiffany Barnes. 2016. Developing a Rubric for a Creative CS Principles Lab. In *Proc. of the 2016 ACM Conf. on Innov. and Tech. in Comp. Sci. Ed. (ITiCSE '16)*. ACM, New York, NY, USA, 290–295.

[4] Domenic V Cicchetti. 1994. Guidelines, criteria, and rules of thumb for evaluating normed and standardized assessment instruments in psychology. *Psychological assessment* 6, 4 (1994), 284.

[5] Steve Cooper, Susan H. Rodger, and et al. 2017. K-12 Teachers Experiences with Computing: A Case Study. In *Proc. of the 2017 ACM Conf. on Innov. and Tech. in Comp. Sci. Ed. (ITiCSE '17)*. ACM, New York, NY, USA, 360–360.

[6] A.L. Delbecq, A.H. Van de Ven, and D.H. Gustafson. 1986. *Group techniques for program planning: a guide to nominal group and Delphi processes.* Green Briar Press, Middleton, WI.

[7] Barbara J. Ericson, Mark Guzdial, and Tom McKlin. 2014. Preparing Secondary Computer Science Teachers Through an Iterative Development Process. In *Proc. of the 9th Workshop in Primary and Secondary Comp. Ed. (WiPSCE '14)*. ACM, New York, NY, USA, 116–119.

[8] David H Gustafson, Ramesh K Shukla, Andre Delbecq, and G William Walster. 1973. A comparative study of differences in subjective likelihood estimates made by individuals, interacting groups, Delphi groups, and nominal groups. *Organizational Behavior and Human Performance* 9, 2 (1973), 280–291.

[9] Aditi Kothiyal, Rwitajit Majumdar, Sahana Murthy, and Sridhar Iyer. 2013. Effect of Think-pair-share in a Large CS1 Class: 83% Sustained Engagement. In *Proc. of the 9th Annual Internat. ACM Conf. on Internat. Comp. Ed. Res. (ICER '13)*. ACM, New York, NY, USA, 137–144.

[10] Sara S. McMillan, Michelle King, and Mary P. Tully. 2016. How to use the nominal group and Delphi techniques. *IJ of Clinical Pharmacy* 38, 3 (01 Jun 2016), 655–662.

[11] Sammuel Messick. 1996. *Techinical Issues in Large-Scale Performance Assessment.* ERIC, Reports-descriptive Validity of Performance Assessments.

[12] Lijun Ni and Mark Guzdial. 2001. Prepare and Support Computer Science (CS) Teachers: Understanding CS Teachers' Professional Identity. In *American Educational Research Association (AERA) Annual Meeting*.

[13] Martijn Stegeman, Erik Barendsen, and S. Smetsers. 2014. Towards an Empirically Validated Model for Assessment of Code Quality. In *Proc. of the 14th Koli Calling Int. Conf. on Comp. Ed. Res. (Koli Calling '14)*. ACM, New York, NY, USA, 99–108.

[14] Martijn Stegeman, Erik Barendsen, and S. Smetsers. 2016. Designing a Rubric for Feedback on Code Quality in Programming Courses. In *Proc. of the 16th Koli Calling Int. Conf. on Comp. Ed. Res. (Koli Calling '16)*. ACM, New York, NY, USA, 160–164.

[15] Alvin Yuan and et al. 2016. Almost an Expert: The Effects of Rubrics and Expertise on Perceived Value of Crowdsourced Design Critiques. In *Proc. of the 19th ACM Conf. on Comp.-Supported Coop. Work & Soc. Comp. (CSCW '16)*. ACM, New York, NY, USA, 1005–1017.