

## ABSTRACT

CATETÉ, VERONICA MEREDITH. A Framework for the Rapid Creation of Quality-Assured Programming Rubrics for New K-12 Computer Science Teachers. (Under the direction of Tiffany Barnes.)

When this research began, AP Computer Science comprised only 0.9% of all AP tests taken in 2014, with roughly 39,000 students [Col97]. After initiatives such as America Competes, CS10K, CSforAll, and the official launch of the new AP CS Principles course in 2016, this number jumped to 2.1% or 104,000 students. This tremendous surge in Computer Science enrollment is a success for the programs, but also reflects the rapid rise in the number of K-12 teachers teaching computer science.

Because of the increasing demand for new CS Principles teachers, they have been recruited from diverse backgrounds. Many CS Principles teachers do not have any background in Computer Science and others have taken just one or two programming courses in college. This lack of experience makes it difficult for new teachers to identify learning goals and provide student feedback on programming lab assignments. With the rapid growth in novice Computer Science Principles teachers, new resources are needed to help teachers not only identify the computational thinking learning objectives in student lab assignments, but also to help teachers grade these programming assignments.

For CS Principles (CSP), several curricula were first designed for the college level and then used as a basis for high school CSP classes. This means that lab assignments were not already annotated with CSP-aligned learning objectives. I sought to provide rubrics that would help new teachers grade the CSP labs and give students feedback on whether they were achieving the CSP learning objectives. I first systematically made rubrics based on common Computer Science grading standards for auto graders and intelligent tutors used in college programming courses. Through this process, I found that the standards used for college Computer Science courses were not detailed enough to assess beginner-level computational thinking in projects. The small portion of the initial rubric focused on learning was biased towards expert programmers as opposed to those still learning the beginning skills taught in CS Principles. Therefore, I reoriented the rubric-making process to be focused on targeting CSP learning objectives.

In order to determine the learning objectives associated with each lab assignment, I applied a Delphi method to poll experts through a controlled group decision making process. Delphi participants generated both associated learning objectives and expected code samples for CSP labs. These were then grouped into topical categories (abstraction, conditional logic, etc.) to develop learning-oriented rubrics. When master CSP teachers and CS undergraduates (novices) used the Delphi-created rubric to grade the Hangman and Brick Wall labs, they achieved a high level of inter-rater reliability. Although the Delphi-created rubrics resulted in consistent grading, the process

to make them was too inefficient to use to create new rubrics for every CSP lab. Instead, I used the lessons learned from the Delphi studies to create a new process based on the Nominal Group Technique using ‘almost experts.’ This new process incorporates both frames of ‘think-pair-share’ and group decision making to expedite the creation process.

Through a series of reliability studies on the initial rubrics, I found that trained Computer Science undergraduates were as reliable as the CS Principles Master Teachers using the rubrics, and could act as surrogate ‘almost experts’ in systematically generating 32 learning-based rubrics using the new methods. I tested these new learning-based rubrics with active CS Principles teachers. I provided teachers high, medium, and low samples of student work, and had teachers mark in code where they were looking for the associated computational thinking concepts. I analyzed the consistency of grade distributions between graders, and also created a visualization to investigate the reliability and usefulness of the rubrics. The visualization helped me verify that teachers using our learning-based rubrics were on the right track for identifying computational thinking in code, but it also revealed that more support is needed for novice CSP teachers to grade code samples that were substantially different from the typical correct solution. These results confirmed that the modified Nominal Group Technique using almost-experts is sufficient to create learning-based rubrics that are reliable for assessing computational thinking in code.

I next applied Baartman’s Wheel of Competency Assessment (WoCA) to further investigate the validity and appropriateness of the learning-based rubrics. Using WoCA, the rubrics can be measured on 12 different quality criteria, including fitness of purpose, cost-effectiveness, meaningfulness, and cognitive complexity. Most of the criteria (10.5 of 12) focus on usability and appropriateness for teachers and the course content. The newly developed learning-based rubrics meet all 10.5 of these teacher-focused WoCA criteria. This analysis shows that the newly created learning-based rubrics are a validated method of support for identifying and understanding learning objectives in student code by novice CS Principles teachers.

This dissertation makes several contributions. It is the first application of the Delphi method for rubric development in K-12 CS education. I also created a novel adaptation of the Nominal Group Technique to address the need for the rapid creation of a large set of learning-based rubrics for computational thinking. I showed that teachers and almost-experts can generally apply the learning-based rubrics to achieve consistent ratings of evidence of computational thinking in student code. Finally, I have performed the first analysis using the Wheel of Competency Assessment to determine the appropriateness of a learning-based rubric for low-stakes assessment for novice student code.

© Copyright 2018 by Veronica Meredith Cateté

All Rights Reserved

A Framework for the Rapid Creation of Quality-Assured Programming Rubrics  
for New K-12 Computer Science Teachers

by  
Veronica Meredith Cateté

A dissertation submitted to the Graduate Faculty of  
North Carolina State University  
in partial fulfillment of the  
requirements for the Degree of  
Doctor of Philosophy

Computer Science

Raleigh, North Carolina

2018

APPROVED BY:

---

James Lester II

---

Aaron Clark

---

Sarah Heckman

---

Tiffany Barnes  
Chair of Advisory Committee

## **DEDICATION**

I dedicate this dissertation to the strong and courageous females who have guided me on my path and encouraged me to pursue my own ambitions. They have taught me that if you don't speak up for what you want, you might end up with a disappointing sandwich made with love.

## **BIOGRAPHY**

The author has a background story so engaging that Oprah Winfrey would want to make it into a Lifetime Original Movie. You should ask her about it one day. Bring the tissues, popcorn, and juice boxes because it's a roller coaster of a ride that's too long to put here.

## ACKNOWLEDGEMENTS

I am grateful to the members of my committee, Dr. Tiffany Barnes, Dr. Aaron Clark, Dr. Sarah Heckman, and Dr. James Lester II for their time, encouragement, and expertise throughout this project.

There are people in everyone's lives who make success both possible and rewarding. My husband, Daniel McMullen, and mother Diana Pasquinelli, steadfastly supported and encouraged me. Daniel, without you I may have well starved to death 30 times over throughout my doctoral studies, thanks for putting food in front of my face.

Additionally I'd like to thank all hands and researchers who have worked on this project, including undergraduates: Lady Kathleen (Wassell) Brennan, Erin Snider, Alex Rouse, Samuel Schoeneberger, Meghana Subramaniam, and Kunj Patel; post-graduate student: David Warren; post-doctoral student: Jen Albert; and CS Principles teachers: Marney Hill and Mark Ruckstuhl.

I'd also like to thank both the SIGCSE and CSTA communities for completing so many surveys and studies and my lab mates for dealing with so many of my frustrations and jublations as told through songs and winded sagas.

Last but not least, I'd like to thank Caroline Law and Carla Bendezu for being awesome mentees who make me proud of them so so often. And also Neil Robson, I taught you computer science as a middle school student, and now you're majoring in it at state, and I'm still here to see it.

This material is based upon work supported by the National Science Foundation Graduate Research Fellowship under Grant No. 61273304 and the Microsoft Research Graduate Women's Scholarship Program.

## TABLE OF CONTENTS

<b>LIST OF TABLES</b> .....	<b>viii</b>
<b>LIST OF FIGURES</b> .....	<b>ix</b>
<b>Chapter 1 Introduction</b> .....	<b>1</b>
1.1 Research .....	2
1.2 Contributions .....	3
<b>Chapter 2 Literature Review</b> .....	<b>5</b>
2.1 CS Principles .....	5
2.1.1 Beauty & Joy of Computing .....	8
2.2 Measuring Computational Thinking .....	11
2.3 Delphi .....	14
2.3.1 Classic Delphi .....	15
2.3.2 Criticisms of Delphi .....	17
2.3.3 Applications of Delphi .....	18
2.4 Discussion on CS Education .....	21
<b>Chapter 3 Study 1: Systematic Rubric Development for CS Principles</b> .....	<b>22</b>
3.1 Introduction .....	23
3.1.1 Brick Wall Assignment .....	24
3.2 Rubric Creation .....	25
3.2.1 Rubrics in CS Ed .....	25
3.2.2 Criteria Selection .....	26
3.2.3 Performance Descriptions .....	26
3.3 Study Design and Methods .....	28
3.3.1 Context and Data Sources .....	28
3.3.2 Methods .....	28
3.4 Results .....	29
3.4.1 Code Examples .....	31
3.4.2 Rubric Refinement .....	32
3.5 Conclusions and Future Work .....	35
<b>Chapter 4 Study 2: Task vs. Learning Based Rubric Evaluation</b> .....	<b>37</b>
4.1 Introduction .....	37
4.2 Background .....	38
4.3 Summer 2015 Delphi .....	39
4.3.1 The Panelists .....	40
4.3.2 Hangman Lab Description .....	40
4.3.3 Survey Rounds .....	41
4.4 Rubric Evaluation Methods .....	42
4.5 Results .....	44
4.5.1 Performance-Based Rubric Results .....	44

4.5.2	Learning-Based Rubric Results . . . . .	45
4.5.3	Between Rubrics . . . . .	47
4.6	Discussion . . . . .	47
4.6.1	Reliability . . . . .	48
4.6.2	Ease of use . . . . .	49
4.6.3	Score distribution . . . . .	50
4.7	Conclusions . . . . .	51
<b>Chapter 5</b>	<b>Study 3: Delphi Methods in CS Principles Rubric Creation . . . . .</b>	<b>53</b>
5.1	Introduction . . . . .	54
5.2	Background . . . . .	54
5.2.1	U.S. K-12 Computing Teachers . . . . .	54
5.2.2	Computational Thinking Rubrics . . . . .	55
5.2.3	Delphi Process . . . . .	55
5.3	Methods . . . . .	56
5.3.1	Lab Assignment Descriptions . . . . .	57
5.3.2	Panelists . . . . .	58
5.3.3	Survey Rounds . . . . .	58
5.3.4	Delphi Application . . . . .	59
5.3.5	Rubric Evaluation . . . . .	61
5.4	Results . . . . .	63
5.5	Discussion . . . . .	64
5.5.1	Delphi Process . . . . .	64
5.5.2	Effectiveness of Rubrics . . . . .	64
5.5.3	Cost-Benefit Analysis of Methods . . . . .	65
5.6	Conclusions and Future Work . . . . .	65
<b>Chapter 6</b>	<b>Study 4: A Streamlined Approach to the Systematic Creation of Rubrics for CS Principles . . . . .</b>	<b>67</b>
6.1	Introduction . . . . .	67
6.2	Background . . . . .	68
6.2.1	Rubrics in CS . . . . .	68
6.2.2	NGT vs. Delphi . . . . .	70
6.3	Methods . . . . .	72
6.3.1	Training new 'Masters' . . . . .	72
6.3.2	Streamlining the Process . . . . .	73
6.3.3	Quality Assurance . . . . .	73
6.4	Results and Discussion . . . . .	74
6.4.1	A Modified NGT . . . . .	74
6.4.2	Rubric Quality and Distribution . . . . .	76
6.4.3	Limitations . . . . .	76
6.5	Conclusions and Future Work . . . . .	77
<b>Chapter 7</b>	<b>Study 5: An Evaluation of BJC Rubrics with Active CS Principles Teachers . . . . .</b>	<b>79</b>
7.1	Introduction . . . . .	79

7.2	Methods	80
7.2.1	Participant Procedure	80
7.2.2	Lab Assignment Descriptions	81
7.3	Results	83
7.3.1	Intra-class correlations	83
7.3.2	Heat Mapping and Visualization Analysis	85
7.4	Discussion	91
7.5	Conclusions	95
<b>Chapter 8 Rubric Development Assessment Using WoCA</b>		<b>96</b>
8.1	Introduction	96
8.2	Background	97
8.2.1	Reliability	97
8.2.2	Validity	98
8.2.3	Low-Stakes Problem-Based Assessments	100
8.3	Measuring Criteria for Assessment	102
8.3.1	Why Rigorously Evaluate Rubrics	102
8.3.2	How to Evaluate Rubrics Meaningfully	102
8.4	Aligning Research to the Wheel of Competency Assessment	103
8.5	Discussion	108
8.6	Conclusions	109
8.7	Future Work	110
<b>Chapter 9 Conclusions</b>		<b>112</b>
9.1	Review	112
9.2	Research Questions	112
9.3	Hypotheses	113
9.4	Contributions	116
9.5	Future Work	117
<b>Bibliography</b>		<b>119</b>
<b>APPENDICES</b>		<b>130</b>
Appendix A	Resources	131
A.1	Beauty and Joy of Computing Lesson Plans	131
Appendix B	Survey Instruments	161
B.1	Study 3 (Delphi) Consent forms and Survey instruments	161
B.2	Study 5 (Rubric Use) Consent forms and Survey instruments	188

## LIST OF TABLES

Table 1.1	The research questions, associated hypotheses, and their locations in this research. . . . .	3
Table 2.1	Differences between AP Computer Science A and Computer Science Principles as noted by the College Board [Col15]. . . . .	6
Table 3.1	Final rubric used for initial grading of student projects in Brick Wall assignment. NOTE: item in italics are additions made to the rubric for clarity. . . . .	27
Table 4.1	Hangman Rubric developed from Learning Objectives and Essential Knowledge selected by Delphi panelists. . . . .	43
Table 4.2	Hangman Rubric developed from performance-based rubric criteria in work by Cateté. . . . .	45
Table 4.3	Descriptive statistics for a comparison of means test on Group 1 vs. Group 2. (In each category Group 1 is the top line.) . . . . .	47
Table 5.1	Summary of metrics used in Delphi Process implementation. . . . .	58
Table 5.2	Sample Parameters category on learning-based rubric for Hangman lab. . . . .	60
Table 5.3	A breakdown of rubric grading metrics. . . . .	62
Table 5.4	Avg. Brick Wall project scores [0-4] using a learning-based rubric. . . . .	63
Table 5.5	Avg. Hangman project scores [0-4] using a learning-based rubric. . . . .	63
Table 6.1	Common learning objectives between between assignments with similar mechanics . . . . .	75
Table 7.1	Participant breakdown for the Spring 2017 study. Participants in the middle row are part of both data sets. . . . .	83
Table 7.2	A sample of the categories and learning goals for the Binary Conversion Rubric. . . . .	86
Table 7.3	A sample of the categories and learning goals for the C-Curve Rubric. . . . .	89
Table 8.1	A summary of processes for meeting Wheel of Competency Criteria . . . . .	103
Table 8.2	Evidence for Wheel of Competency Assessment pt. I . . . . .	105
Table 8.3	Evidence for Wheel of Competency Assessment pt. II . . . . .	106
Table 8.4	Evidence for Wheel of Competency Assessment pt. III . . . . .	108

## LIST OF FIGURES

Figure 2.1	(a) Computational Thinking Pattern Spiral shows concepts from the simple to the complex (b) Comparison of CTP Graphs: Sim-Sokoban combination . . .	13
Figure 2.2	A traditional approach to carrying out the classic Delphi method. . . . .	15
Figure 3.1	Levels of abstraction for the Brick Wall assignment . . . . .	24
Figure 3.2	Rubric scores for group 1 and group 2 . . . . .	30
Figure 3.3	Low-level example, scored as: Accuracy = 3, Efficiency = 2, Reasoning = 1, and Readability = 1. . . . .	31
Figure 3.4	Medium-level example, scored as: Accuracy = 4, Efficiency = 3, Reasoning = 2, and Readability = 3. This program shows lack of loops and logic . . . . .	32
Figure 3.5	High-level example, scored as: Accuracy = 4, Efficiency = 4, Reasoning = 4, and Readability = 3. Program uses loops & math to generate correct output. . . . .	33
Figure 3.6	Two brick walls with differing row alignment on the right side. . . . .	34
Figure 3.7	Blocks showing student misunderstandings. . . . .	34
Figure 4.1	(a) New block to display a secret word ‘Hangman’ style. (b) Beginning structure of a ‘Word-list’ block. . . . .	41
Figure 4.2	This figure compares projects evaluated by a performance-based rubric between group 1 and group 2. . . . .	46
Figure 4.3	This figure compares projects evaluated by a learning-based rubric between group 1 and group 2. . . . .	48
Figure 4.4	Student score correlations across rubrics . . . . .	49
Figure 5.1	A simplified version of the Delphi Process. . . . .	56
Figure 5.2	Levels of abstraction for the Brick Wall assignment . . . . .	57
Figure 5.3	Block displays a secret word ‘Hangman’ style. . . . .	57
Figure 5.4	A portion of the code sample critiqued by Delphi Panelists in Round 3. . . . .	60
Figure 6.1	(a) The Delphi Process hinges on iterative cycles of survey generation and response aggregation. (b) The Nominal Group Technique encourages participants to write ideas down individually prior to sharing with the group. . . . .	71
Figure 6.2	Our Modified Nominal Group Technique pairs participants prior to group discussion. . . . .	74
Figure 7.1	Base Snap! code for a Decimal to Binary conversion block. . . . .	81
Figure 7.2	Level 1 and 2 of the C-Curve algorithm . . . . .	82
Figure 7.3	Extended recursion levels of the C-Curve . . . . .	82
Figure 7.4	Heat map of areas that influenced grader decision for ‘Mathematics’ in Binary Conversion 1 . . . . .	87
Figure 7.5	Heat map of areas that influenced grader decision for ‘Mathematics’ in Binary Conversion 2 . . . . .	87
Figure 7.6	Heat map of areas that influenced grader decision for ‘Mathematics’ in Binary Conversion 3 . . . . .	88

Figure 7.7	Heat map of areas that influenced grader decision for ‘Mathematics’ in C-Curve Sample 1 . . . . .	90
Figure 7.8	Heat map of areas that influenced grader decision for ‘Mathematics’ in C-Curve Sample 2 . . . . .	90
Figure 7.9	Heat map of areas that influenced grader decision for ‘Mathematics’ in C-Curve Sample 3 . . . . .	91
Figure 7.10	Combined score distribution for C-Cure Sample 2, ICC(2,14)=.75. . . . .	92
Figure 7.11	Teacher score distribution for C-Cure Sample 2, ICC(2, 6)=.74. . . . .	92
Figure 7.12	Student score distribution for C-Cure Sample 2, (ICC2, 8)=.31. . . . .	93
Figure 8.1	Baartman’s Adapted Framework: The Wheel of Competency Assessment . . . .	100

## CHAPTER

# 1

## INTRODUCTION

In 2010, six different pilot courses were proposed for the new Advanced Placement Computer Science Principles course being developed by the College Board. One such proposal was to refit the college-level Beauty and Joy of Computing course into the CS Principles framework. Both courses were designed using the 7 Big Ideas of computing (Abstraction, Creativity, Algorithms, Programming, the Internet, Big Data, and Social Impacts of Computing) so relatively minor adjustments had to be made to align to the new framework. Unfortunately, throughout the initial pilot years, the CS Principles framework as well as the BJC curriculum underwent numerous changes. This created a situation where teachers participating in the CS Principles (BJC) pilot program were unsure of the intended learning objectives of the BJC labs. Several of the original BJC assignments were more rigorous than what was required for AP CS Principles so teachers were less prepared to teach them or provide sufficient student feedback for them. Furthermore, as BJC was originally a college course, required documentation for K-12 school systems didn't exist. This includes things like pacing guides, blueprints, and other documents linking the course materials back to the national standards in the AP CS Principles Framework.

CS Principles teachers have numerous backgrounds ranging from the more typical business and math to Spanish language and chemistry. Many do not have a background in Computer Science and might have taken just a few programming courses in college. This lack of experience made

it difficult for teachers to identify what learning goals were associated with each lab assignment. Those who originally made the lab assignments had moved on or were no longer available to work on the project. While the pilots for CS Principles were occurring, there was another national movement, CS10K, where Universities and other groups were working to train 10,000 new high school computing teachers. This eminent growth in novice Computer Science Principles teachers, meant that new resources would have to be built to help teachers not only identify the computational learning objectives in student lab assignments, but also to help teacher grade these programming assignments. As one of our newer BJC professional development attendees stated, “I’m not sure what my students are supposed to be learning from this lab or even how to tell if they did.”

There are over 42,000 high schools in the United States and despite the critical and growing importance of computer science, only 2,100 of them are certified to teach Advanced Placement<sup>1</sup> Computer Science A (AP Computer Science). When this research was first proposed, AP Computer Science comprised only 0.9% of all AP tests taken in 2014, roughly 39,000 students [Col97]. After initiatives such as America Competes, CS10K, CSforAll, and the official launch of the new AP CS Principles course in 2016, this number jumped to 2.1% or 104,000 students. This tremendous surge in computer science enrollment is a success for the programs, but also reflects how many new computing teachers are entering the fold and teaching AP level courses.

## 1.1 Research

Due to the increased need for computer scientists and the unprecedented growth of CS Principles, I have decided to focus on how we can help novice CS Principles teachers identify student learning of computational thinking through coding assignments by investigating the following set of questions listed in Table 1.1. My overriding research goal is to identify and clarify core lessons of the CS Principles curriculum, specifically in BJC, in order to provide teachers with a rubric system to evaluate student artifacts for evidence of achieving the computational learning objectives.

I begin by first creating rubrics using methods from auto graders and intelligent tutoring systems (Chapter 3). I then refactor the rubric design to use learning objectives established by an expert Delphi panel and compare it to the initial rubric design (Chapter 4). I compare a more traditional Delphi method to a modified version that is more time efficient (Chapter 5). The lessons from that study, I then apply to a more streamlined process in Chapter 6 in order to create the full set of rubrics for BJC teachers. In the final study, I analyze those rubrics for reliability and teacher helpfulness in their ability to identify computational learning objectives in code (Chapter 7). I conduct an overall level of quality analysis in Chapter 8 to conclude this research. My contributions are listed briefly in

---

<sup>1</sup>Advanced Placement courses offer students college credit for taking higher level courses in high school.

the next section. Then Chapter 2 presents relevant background literature. Those familiar with my proposal may wish to start at Chapter 5.

**Table 1.1** The research questions, associated hypotheses, and their locations in this research.

Research Questions & Hypotheses	Chapter
How can we help teachers better understand learning objectives for labs and identify whether they are achieved in student artifacts?	
[H1] We can use the Delphi method to create learning-based rubrics that perform better than traditional task-based rubrics at helping teachers grade assignments meaningfully. [H2] Use of the Delphi method to create rubrics will lead to sufficient levels of inter-rater reliability among novice graders for low-stakes assessments such as lab programming projects.	Chapter 4; Chapter 5
How can we modify educational psychology methods to meet our needs for creating a large set of rubrics for classroom settings?	
[H3] We can use a modified NGT approach to efficiently and quickly produce a full suite of quality assured rubrics for BJC lab assignments. [H5] We can apply the Wheel of Competency Assessment to show that the created rubrics are appropriate for low-stakes assessment of CS Principles labs.	Chapter 6 & 7; Chapter 8
How well do teachers identify computational thinking in student artifacts using task-based and learning-based rubrics?	
[H4] Learning-based rubrics will help support beginning CS Principles teachers consistently assess important computational thinking elements in student code.	Chapter 7

## 1.2 Contributions

1. I conducted the first application of the Delphi Method to create content-validated rubrics for a K-12 computing course. Using these methods we were able to gain insights on streamlining a robust process for creating a large quantity of quality rubrics, which benefits the rapidly expanding CS Principles program and can be applied to other newly developed novice computing courses.
2. I redesigned the rubric creation process to be more cost-effective and thus feasible to replicate for use in real-life applications and school settings. This allows for quick distribution of rubrics on rapidly changing courses. As computing trickles down the K-12 pipeline, we will be presented with more teachers from diverse backgrounds and course content that will have to

adjust as students in lower grades become more experienced. Thus, being able to generate new reliable rubrics quickly and so that they support novice teachers will be beneficial.

3. I proved that the rubrics created with the new modified NGT process and almost-experts are as reliable as the initial Delphi generated rubrics. As our modified NGT method was more cost-effective and efficient due to using surrogate experts, it was critical to ensure that they held up to the same standards as the initial Delphi rubrics. Ensuring a level of quality assessment to our rubrics is important for their adoption by teachers.
4. I devised a novel application of the Baartman's Wheel of Competency Assessment to establish a high level of validity, value and acceptability of the rubrics. By aligning our rubrics to the Wheel of Assessment we were able evaluate whether these rubrics met their intended goals. As we used new tools and open-ended processes to assess student work, it is important to evaluate their appropriateness for use by new CS Principles teachers.
5. I led a junior research team composed of upper classification computer science undergraduates in generating 32 Beauty & Joy of Computing rubrics designed to measure implementation of learning objectives in student lab assignments. This is the first such system, as well as the first such system specifically designed for beginning teachers.

## CHAPTER

# 2

## LITERATURE REVIEW

As Jeannette Wing first posed in 2006, “computational thinking involves solving problems, designing systems, and understanding human behavior, by drawing on the concepts fundamental to Computer Science [Win06].” The National Research Council (NRC) believes that computational thinking is a skill necessary for all persons in every field. When they met to discuss the scope and nature of computational thinking in the classroom, leaders deliberated on how we recognize computational thinking in students and their work [Nat10]? As a part of this research, I intend to help answer this question. In order to do so, I draw on frameworks and methods from Computer Science and Education literature. Below I present background information on CS Principles and the Beauty and Joy of Computing followed by measures of computational thinking and approaches to conducting Delphi studies.

### **2.1 CS Principles**

With the introduction of CS Principles, there has been some resistance concerned with its replacement or duplication of AP Computer Science, a course with historically low turnout and diversity

[ACM15]. In fact, AP Computer Science A was the least diverse AP course <sup>1</sup> from 2010-2014 with 85% male and 84% Asian or Caucasian students [Col97]. Advocates for CS Principles reiterate that CS Principles is not a replacement and is instead a complement to the original AP Computer Science course. The primary difference between the two courses is that AP Computer Science is an applications course with over 20 hours of hands on lab and Java programming assignments recommended, whereas CS Principles has programming as only a small portion of the class focus, and even then, the programming is typically done in a non-syntax specific manner (e.g. Snap!, AppInventor etc). In CS Principles, the creators want to bring home the global impacts and widespread influence of computing technologies. See Table 2.1 below for key difference between the two courses.

**Table 2.1** Differences between AP Computer Science A and Computer Science Principles as noted by the College Board [Col15].

<b>AP Computer Science A</b>	<b>AP Computer Science Principles</b>
Curriculum is focused on object oriented programming and problem solving	Curriculum is built around fundamentals of computing including problem solving, working with data, understanding the Internet, cyber security, and programming
Java is the designated programming language	Teachers choose the programming language(s)
Encourages skill development among students considering a career in computer science or other STEM fields	Encourages a broader participation in the study of computer science and other STEM fields, including AP Computer Science
AP assessment experience: Multiple-choice and free-response questions (written exam)	AP assessment experience: Two performance tasks students complete during the course to demonstrate the skills they have developed (administered by the teacher; students submit digital artifacts) Multiple-choice questions (written exam)

Researchers have found that the disconnection between computer programming in AP Computer Science and its implications for people and everyday lives has impacted the overall representation of females interested in taking the course [Col97]. We see from other STEM courses that females are interested and capable in Math, Biology, and other science courses [Mod12]. Women have the same capability to take a programming course, however they don't see how it relates to or impacts societal good. Based on this evidence, the NSF and College Board paired up to create a computing principles course that focuses on the people and societal impacts of computer science.

<sup>1</sup>This statistic does not include the four culture courses (German, Italian, Chinese, or Japanese), which typically have a low turnout.

The CS10K project mentioned previously is focused on two new courses: Exploring Computer Science (ECS) and Computer Science Principles. Both courses focus on making computing relevant to underrepresented minorities; however, their benefits are slightly different. AP courses are often the only high school courses that offer college credit, which make them attractive to students and schools. They also offer a point of national leverage and hold students to the same standard of final exam. ECS is important for those who might be intimidated by an AP course. ECS offers a broad range of ideas and counts towards college prep and Career and Technical Education credit (CTE).

One of computer science education's landmark publications is the book *Stuck in the Shallow End* (SSE), by Margolis, which discusses the racial inequities in the computing classroom. This book is the culmination of several years of research on the Los Angeles Unified School District (LAUSD). The findings included a lack of qualified teachers and minimal community support for CS teachers in low-income schools [Mar08].

Three case studies are detailed in SSE; the first study is in a lower income school with a predominantly Latino and English learner population. This school had computer labs and equipment available, although the only computer programming course stopped being offered during the third year of the study. The promotional videos showcasing the school's status as a Digital High School show students using computers for courses like art and design. The programming teacher was primarily a mathematics teacher who was self taught in computing. The students copied programs from a projector and wrote them into a computer; rarely would an activity or assignment go beyond calculations. Furthermore, when there was a class being offered, not many minorities were in it, as guidance counselors would relegate these students to the vocational wing. The minority students were often seen as lacking the ability to do the higher level thinking required for computing. It was the view of administration that since many students did not know English well enough, it was foolish to teach them higher level courses.

Secondly, in a predominantly African American school, of lower middle class level, computing courses were again being taught on a basic level; higher computational thinking concepts were not being utilized. This school had a group of students who self-identified as "techies" who would go around helping the administration keep computing labs up to date and fixing technical problems. The computing labs were often off limits and did not have Internet access at each machine. This school was also failing the mathematics end of grade exams and had low marks in the No Child Left Behind program. In this case, the administration did not want to sacrifice a teacher to computer science, a course that isn't required for graduation, when a more critical math class could be taught. There was a third affluent school that was a part of the Digital High School program, which did have well supported technology and computing programs. However, the school students and counselors themselves promoted a sense of culture around CS as being 'white' and for a particular kind of person;

minorities and girls felt excluded. So even though the students had access to quality programs, they did not want to take part because of what their peers might think, or how their counselors viewed them.

This research gave way to the enhancements of teacher training and enrichment to AP Computer Science. Unfortunately, for many of the students, this barrier was too high for those lacking prior CS knowledge or experience. ECS was the answer to the gap in LA and other similar areas. It targets 9th and 10th grade mostly (although some use it in middle school or higher grades) and includes the cultural aspects of students in the course. ECS also has a very tight knit community of teachers and teacher training available. In order to ensure the same quality of teaching and student experience, ECS also features a very strict curriculum (albeit an engaging one). Researchers still wanted to produce a high enough level computing course to count as AP while addressing the similar diversity problems. CS Principles was created to be both informative enough and flexible enough to work as an AP credit.

CS Principles is a course framework for teaching the principles of computer science. The creators of this framework identified seven big ideas central to computing. In addition to programming; abstraction, algorithms, and social impact are also central to this course. It is important for students in CS to not only know about syntax while programming, but also to focus on the bigger picture, such as using abstraction to simplify problems and break things down into sensible pieces. The creators want students to be able to abstract algorithms and apply them to different areas. An example of this is researchers at Microsoft using the same kind of algorithms for detecting spam mail to help identify bad blood cells for those with HIV [Cla11]. Being able to make connections between ‘boring’ tasks and life saving techniques is a brilliant way to attract a more diverse demographic into computing. A research study on women and career perspectives, found that women want careers that help people [Mod12; Cec09; Sch08]. Highlighting the global impact of computing can make computer science a viable option for those who want to focus on projects helping people. The AP exam for CS Principles has also been modified compared to other standard exams. Students turn in a Performance Task portfolio that contains both collaborative and individual projects. Projects entail both written response and creation of a digital artifact. The performance tasks are designed to allow students to have broad latitude in personally selecting their focus and topics.

### **2.1.1 Beauty & Joy of Computing**

Since CS Principles is a framework for a course, there are many different implementations of it. Ralph Morelli, a professor at Trinity College who joined during the Phase II pilots in 2012, uses a mobile computing approach to attract students to CS Principle. Similarly, Kelvin Sung a professor at

the University of Washington Bothell, who joined CS Principles after working with Larry Snyder (a Phase 1 CS Principles host), started a digital thinking course focused on game development [Mor13; SS14]. BJC on the other hand uses a syntax free language called Snap! as its main vehicle; it also incorporates mobile apps, and game development. Each course has been implemented with specific goals in mind and has separate strengths. I will discuss a few of these and then give an in depth look into the BJC curriculum.

Each of the CS Principles implementations revolves around the 7 big ideas of computation: creativity, abstraction, algorithms, programming, the Internet, big data and global impact. What distinguishes the implementations is how much time they devote to each topic, and what tools they use to do so. In the first phase of CS Principles pilots, coverage of *algorithms* ranged from 13% to 29% [Sny12]. In the case of our first CS Principles example, Mobile CSP by Morelli, the core programming aspects of the class are done through MIT's AppInventor technology. Mobile CSP is broken into 7 units, the first being a setup and pre-course unit, the others revolving around mobile applications and creativity. These units are: Mobile computers and mobile apps; Graphics and Drawing; Animation, Simulation, and Modeling; Algorithms and Procedural Abstraction; Lists, Databases, Data and Information; The Internet [Tri15]. The first three units utilize AppInventor tutorials and activities from CS Unplugged [Can15]. For the data section AppInventor allows students to use built-in database features highlighting tagging, persistent data, and shared data. Mobile CSP features 32 CS Principles (or non-programming) lessons and 28 programming lessons. Morelli offers a miniature Massive Open Online Classroom (MOOC) to train teachers in using MIT's AppInventor software and allows teachers to pick and choose which modules they apply to their course [Mor15]. In Mobile CSP students learn computer science by building socially useful mobile apps.

Similar to Mobile CSP, Kelvin Sung's Digital Thinking course is also themed, this time around video game development. The implementation at University Washington-Bothell (UWB) is delivered as a university-freshman level, disciplinary overview. The developers of the course wanted to focus on giving students an 'authentic' programming experience, i.e. a text-based language. C# and an XNA game-development plugin for Visual Studio was chosen as the main outlet for programming. The course is adapted from a CS1 game course which utilized an XNACS1 library as an extra layer of abstraction onto C# programming. Students only needed to learn two main functions (InitializeWorld() and UpdateWorld()) and one object (XNACS1Rectangle) to work with the library [SS14]. Of the 33 students in the course, just two failed to show up to the final exam and were unsuccessful in the course. The Digital Thinking course showed that they could successfully teach students CS Principles while using a text based language approach. The other aspects of the course include its 11 week lecture series covering topics such as Digitization and the CD ROM; Color + Binary Numbers; Privacy and Ethics; Intro to Processing; Processing with Variables; Data Types and Functions; Loops

and Conditionals; Computational Complexity; AI; Instruction Execution; Networking and DNS; and finally WWW: Search and Tagging. This course focuses much more on the lower level details of computing and digitization than other implementations like Mobile CSP.

BJC aligns with Mobile CSP in that it uses a graphical language approach and uses *Blown to Bits* as a reading companion for the class. *Blown to Bits: Your Life, Liberty, and Happiness After the Digital Explosion* gives a balanced perspective on personal data and the digital age. A large portion of the book covers ethics in computing and the dangers of having information stored on the Internet, some by choice others not. Both the high school teachers and students find this book very intriguing and socially relevant. Unlike a traditional textbook, this one is written as pop fiction and is meant to engage the reader. BJC teachers have been very quick to pick up the literature and are able to facilitate discussions around the different chapters. The book does a good job at relating to topics the students are learning like bits and binary to information or technology they use every day.

BJC emphasizes reflection on the impacts of computing on society because the designers want students to be critical thinkers, understanding the (often unintended) social, ethical, and economic implications of technology and computing. In each implementation of BJC, current events and interdisciplinary applications are emphasized to connect with students and maintain relevance. The latest version of the course offers 20 topical units, the first fifteen being organized by programming concept, and the remaining five as extensions (e.g. Simulation in Science) [Uni15]. The programming units are accompanied by additional videos and resources on a non-programming topic such as ethics in computing or social implications of computing. What separates BJC is its inclusion of recursion and higher order functions. From the program's funding grant, "BJC invokes passion, beauty, joy, and awe through engaging students in a rigorous computing curriculum that promotes creativity and collaboration using Snap!'s visually rich programming environment, while also provoking thought around current events and how computing relates to people's lives" [Gar11]. The researchers believe that recursion and higher order functions (HOF) exemplify this beauty of abstraction, one of the core ideas in computer science and in the CS Principles course design.

A higher order function is one whose domain or range includes other functions [AS96]. Higher order functions are an extremely powerful means to capture patterns of control flow in a program, eliminating the need to keep track of index variables and individual array elements. In order to successfully teach recursion and HOF the researchers built the Snap! programming language to support these implementations [Har14]. An example of this is using functions as data in the "map over" block where a user can map an operation like  $5 \times$  over a list resulting in each element of that list being multiplied by five. Other examples include storing a list of procedures that can be iterated through and executed as needed [Uni15]. Although these types of applications are generally considered complex and left until a CS2 course [Eng15], the simplistic interface of Snap!

allows students to implement and grapple with the ideas of these at a much earlier point. Where as Digital Thinking focused on an authentic programming environment, BJC focuses on authentic programming problems.

With many of our CS Principle teachers being new to the curriculum or even subject, we must be able to support them and help them identify what students are learning, how they are learning, and ways in which we can facilitate teachers teaching. Shulman suggests that teachers should have a strong background in three main areas: subject matter, curriculum knowledge, and pedagogical experience [Shu86]. Teachers can go to professional development training for curriculum and have pedagogical experience already. The main part where CS educators can help is in the subject matter. Without strong preparation in the subject matter, teachers are consistently just a step ahead of their students. Many teachers can get flustered when students ask questions that they don't know the answers to [CST08]. Gal-Ezer recommends that teachers need to have the computing skills, plus the ability to convey it to kids [GES10]. It's difficult to convey knowledge at a lower level when you don't know the knowledge to begin with.

## 2.2 Measuring Computational Thinking

As researchers have begun exploring the space of computational thinking (CT), the definition has morphed; specifically to include "formulating problems so their solutions can be represented as computational steps and algorithms" [Aho12]. From a recent survey of the field, Shuchi Grover and Roy Pea compiled a list of elements that are now widely accepted by the academic community as comprising computational thinking:

Abstractions and pattern generalizations (including models and simulations)	Structured problem decomposition
Systematic processing of information	Iterative, recursive, and parallel thinking
Symbol systems and representations	Conditional logic
Algorithmic notions of flow of control	Efficiency and performance constraints
	Debugging and systematic error detection

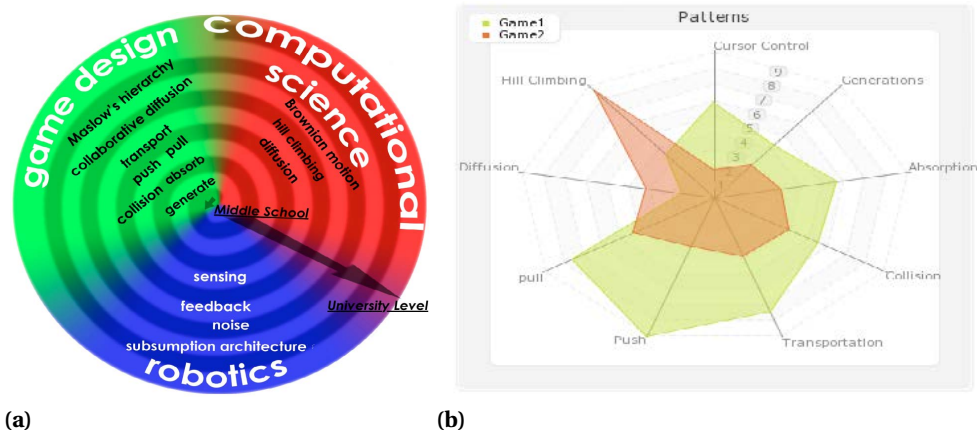
Brennan and Resnick break computational thinking down into three areas: concepts, practices, and perspectives [BR12]. Concepts are the elements of programming such as conditionals and parallel threads. Practices are performing tasks including debugging and iterative design. Lastly, they include perspectives, which is the more abstract area in his breakdown. Computational thinking perspectives, as described by Brennan and Resnick, are demonstrated through how students form thoughts about the world around them and about themselves. One can do this by making

connections between other concepts or applications, by expressing themselves through computing or asking questions [BR12]. Pulling code from children in the Scratch community, they used the Scrape tool [Wol11] to analyze code for frequencies of block use. They compared a new member's project (1 week in community) to a more active member's project (3 year member, 49 projects). Results show there is a clear distinction in number and type of blocks used, demonstrating the differences in understood CT concepts. However, their results were limited as they were unable to identify practices or perceptions of the designers. The researchers followed up with in-person interviews and asked kids what parts of their code did. The researchers were surprised however, to find out that the kids did not always know for sure. In one case, researchers were interviewing a student about his project, and the kid told them that he borrowed the code from someone else because the results looked cool. From this the researchers found that presence does not equal deep understanding. They also asked about practices, however the kids claimed to never get stuck and had lapses in memory. The third stage of research for studying computational thinking involved a classroom setup with design scenarios. Students in class were given external programs over the course of the semester and were told that another student needed help fixing his program. From this, researchers could get a more accurate report on strategies and practices involved, but again learning about perspectives was difficult. Limited observations did not provide the appropriate level of insight into the thought formation process of the students.

In Brennan and Resnick's research, they were trying to develop a framework for studying the development of computational thinking. One of the difficulties they had in analyzing Scratch projects was the projects from Scratch Online that were remixed from others. They were unaware of the students' lack of knowledge. In Snap! projects, students do not have the ability to remix other works, which alleviates the problem of copy-paste. Students would have to understand what is going on before they finish. Therefore, if the students received directions for a project instead of a nearly completed project, they would be able to demonstrate CT concepts.

Further work by Koh attempts to develop a method for the automatic recognition of computational thinking. In Koh's 2010 paper, they use a computational thinking spiral; see Figure 2.1 (a), developed for the Scalable Game Design course to analyze what computational thinking patterns are in students' code. When analyzing the student project (written using AgentSheets) the researchers look at program behavior similarity as well as use latent semantic analysis (LSA) to evaluate student code. They illustrate the frequency and types of critical thinking patterns implemented in their Computational Thinking Pattern (CTP) Graph (a star plot with CT's listed as the outer vertices), Figure 2.1 (b).

The authors suggest that the CTP Graph could potentially demonstrate the existence of knowledge transfer, not just in CS but also in other fields by layering a student's projects over time [Koh10].



**Figure 2.1** (a) Computational Thinking Pattern Spiral shows concepts from the simple to the complex (b) Comparison of CTP Graphs: Sim-Sokoban combination

The authors hope that use of the graph will allow reviewers to more easily see the degree to which concepts are present. Current limitations include, as Koh states, the “arbitrary nature” of the specified computational thinking patterns used in the graph as well as the number of them. The patterns specified in the CT spiral include items such as diffusion and hill climbing that aren’t necessarily generalizable concepts. The researchers also had difficulty in differentiating the concepts. We agree that the CTP graph could be useful for an at-a-glance overview of particular concepts that show up in students’ code.

In the same spirit of a rubric for evaluation, Concept Inventories (CI) are being used by STEM fields to measure misconceptions and conceptual understandings in a topic across instructors, institutions, and pedagogical practices. A concept inventory is a standardized assessment tool designed to measure student understanding of the core concepts of a topic [Gol10]. These are designed to be metrics that evaluate the overarching fundamental concepts and not an exhaustive final examination. The purpose is to improve pedagogy, rather than critically assess students [AW11]. The generality of the CI is what gives it such broad range of application across institutional domains.

One of the most used concept inventories to be developed is the Force Concept Inventory (FCI) used by Physics, developed in 1992 [Hes92]. Repeated use of the FCI enlightened teachers to the misconceptions that students had about force, even after significant instruction [CM01]. Results of the FCI led to a great reform in Physics education including use of Peer Instruction [CM01]. Hake ran a study with 6000+ students in 62 different courses, and found that in each traditional course

there were lower learning gains<sup>2</sup> ( $NG = 0.3$ ) compared to 85% of interactive-engagement courses ( $0.3 < NG < 0.7$ ) [Hak98]. The strength of this study was that it was able to use the FCI (now a gold standard for other CIs) as a common measure among institutions and instructors.

Taylor performed an extensive literature review in 2014 investigating the development, use, and difficulties of CIs for Computer Science [Tay14]. Only two validated CIs for Computer Science exist. They are both at the University level, one for Digital Logic created in 2010 [HH10], and another for CS1 (language independent) created in 2011 [TG11]. The CS1 concept inventory is actually described as a computational assessment, and question answers do not follow the typical creation process as other CIs, but is nonetheless valid via mass distribution testing. Other unvalidated or incomplete CIs have been created for Operating systems [WT14], Computer architecture [Por13], Algorithms and Data structures [PV13], and Binary Search Trees and Hash tables [KW14].

The typical process for creating a concept inventory is very lengthy and involves determining the topics, identifying student thinking, creating open-ended survey questions, creating forced-answer tests, validating test questions through interviews with experts, followed by administering and statistically analyzing the inventory [AW11]. The importance of identifying student thinking is to add valuable distractor answers to multiple choice questions. The distractor draws students with misconceptions toward the associated distractor, which gives test administrators (teachers) insight into which concepts and misconceptions students are struggling with.

Other common practices for developing a CI include using a Delphi Process to facilitate reaching a consensus among CI developers for key concepts [Gol08]. Taylor discusses difficulties unique to the computing discipline. One of the main arguments is that, unlike the older fields of Calculus and Newtonian mechanics, Computer Science is changing rapidly. The time and effort being used in other STEM fields is worth it because those fields are not as likely to change in the next 20 years. BJC and CS Principles teachers are requesting materials to identify learning objectives, for themselves, in course materials. They are unclear of the current learning goals in the programming assignments, and would benefit from a rubric letting them know what a particular concept looks like in code. As a concept inventory asks coding questions that reveal particular understandings, the rubric would reveal code that applies to known objectives. In order to proceed with this development of a rubric, we can use similar practices to those used for concept inventories, particularly Delphi Processes.

## 2.3 Delphi

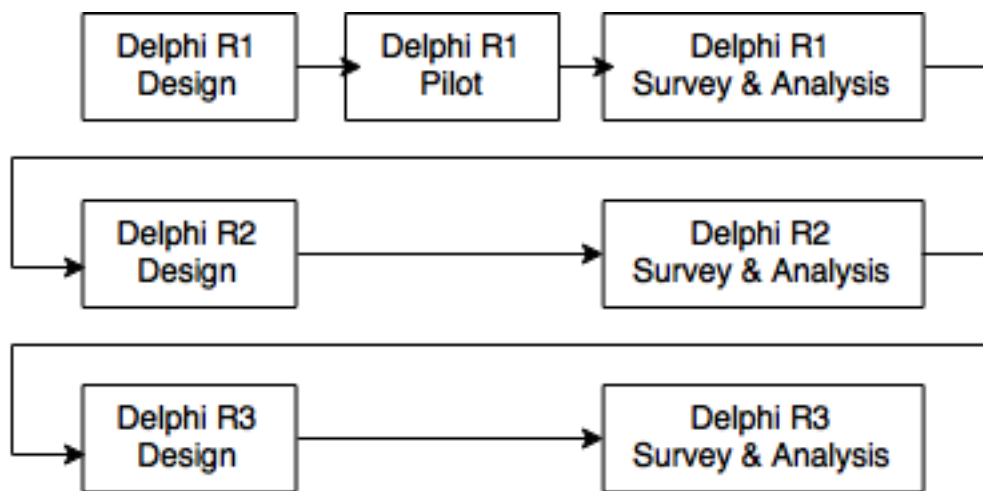
The Delphi process is a technique originating in the 1950s to obtain the most reliable consensus of opinion from a group of experts using intensive questionnaires interspersed with controlled opinion

---

<sup>2</sup>Normalized Gains (NG) were used to account for initial student knowledge

feedback [DH63]. Instead of relying on random samples, Delphi studies represent the best thinking and opinions of a group of people chosen for their special knowledge and experience [AZ96; Lec84]. As emphasized by Adler, expertise is not tied to having a Ph.D. in the subject, but rather knowledge and practical engagement with issues under investigation [AZ96].

The Classical Delphi is characterized by four pillars: anonymity of participants, iteration for participants to refine their views, controlled feedback informing participants of other perspectives, and statistical aggregation of the group response to allow for quantitative analysis and interpretation of data [RW99]. As part of the Delphi Process, panelists complete a series of questionnaires which are analyzed and processed, with the results integrated into the next stage of survey rounds; Figure 2.2 outlines this process.



**Figure 2.2** A traditional approach to carrying out the classic Delphi method.

As outlined by Skulmoski, the typical Delphi method can vary greatly in application [Sku07]. The modified Delphi comes in many forms and has evolved over the last few decades. We'll explain the Classical and some of these modified Delphi the subsection below.

### 2.3.1 Classic Delphi

Adler and Zigler pose that one should use a Delphi study to help make informed decisions when the problem has no monitored history nor adequate information on future/present development and for problems that require various perspectives [AZ96]. To its core the Delphi has two main phases, 1. knowledge generation 2. knowledge evaluation.

Project Delphi was originally used in the early 1950s by the U.S. Air Force to apply expert opinion to the estimation of bombing requirements to reduce enemy munitions output [DH63]. In this process, experts were identified and independently answered a series of questions eliciting the creation of recommendations. Researchers then had a moderator team collate the responses, and give the information back to the experts (again independently) so that they could narrow down their responses and reach an agreement. The researchers used a moderator team as a go between to help eliminate researcher bias in the coding of responses in the study.

In the 1970's Linstone and Turoff used the Delphi method to make policy judgments. They valued the Delphi as a tool for policy type questions that involved aspects such as goal formation for which there were no overall experts, only advocates and referees. For these policy questions, the resolutions must take into consideration the conflicting goals and values espoused by various interest groups as well as facts and staff analysis [LT75]. Linstone and Turoff use the Delphi to help stakeholders formulate ideas, expose options, determine positions, explore disagreement, and evaluate underlying reasons all while maintaining an objective perspective. They found 3 rounds to be sufficient in attaining stability in responses. When analyzing Delphis they found that the most common judgment indicators were simple rank and likert type questions. Linstone and Turoff also set the standard for consensus to be when the interquartile range is no larger than two units on a ten unit scale.

By the 1980's the Delphi method had started being used for futures predictions. In order to make informed decisions on policies and funding that would impact the mentally ill for the next 20 years, Lecklitner surveyed 345 persons that played the most prominent roles in decisions regarding the rights of the mentally ill. In this situation, stakeholders now included current and former mental health care recipients, parents and family, mental health and social service providers as well as patient rights advocates, community representatives, and academics and researchers in related areas. In Lecklitner's study the samples represented the best thinking and opinions of a group of people chosen for their special knowledge and experience [Lec84]. Only two rounds were conducted in this study, the first to generate forecasts on quality of care and concerns that the group had. The second round to generate plausible solutions to these issues. The main goal was not to come to a single consensus, but rather to get people talking about and making informed decisions about the future.

The systematic process of opinion gathering that is the classical Delphi has been outlined by Brooks in the following eight steps [Bro79]:

1. Panel of experts is identified.
2. The willingness of the individuals to participate is determined.

3. Individual input on a given issue is gathered and combined into basic statements.
4. The data provided by the panel are analyzed (by a moderator team).
5. The assembled group input (questionnaire) is mailed to each panel member for assessment.
6. The new input is analyzed (by the moderators). The results, indicating the distribution of responses, are returned to the panel.
7. Each participant is asked to examine the data and to reassess his own position based on the group's responses. A participant whose personal position varies significantly from the group norm is asked to provide a rationale to support the divergent view. The length of the rationale (remarks) is limited to keep responses brief.
8. The input is analyzed by the researcher. The input is shared, in addition to the minority supporting statements, with the panel. Each member is asked again to review his position; and, if still not within a specified range, to support that position with a brief rationale.

From the early days of the Delphi, the process has begun to morph in new ways to accommodate the growth in diverse sets of research questions. One variation to the classical Delphi has been scope of response in the first round. In Friend's research, panelists were given an initial set of narrow questions based on literature derived content instead of generating the initial statements or opinions to a problem [Fri01]. This is further expanded in Cabaniss' study on computer-related technology use by counselors, Delphi panelists were tasked with submitting written examples of CRT use by counseling professionals [Cab01]. This deviated from the traditional survey format of Likert type questions, providing a richer context of understanding to the researchers.

### **2.3.2 Criticisms of Delphi**

Since its inception, the Delphi model has drawn criticism and support. The primary source of criticism comes from a report from RAND researcher Harold Sackman who identified perceived failings of the method [Sac74]. Many scholars have built on this work either using it to refute or expand further criticism. In general, criticism of the method falls along three lines; those relating to the role of the researcher, those relating to the role of the experts, and those related to the process itself.

The criticism with a researcher using Delphi starts before the process begins as there are no clear guidelines on how to employ the method [LT75]. Researchers can confound the process on start by using vague initial questions [Sac74] and researchers engaged in the method suffer from "hurry and wait syndrome" needing to rush data analysis in order to continue the process without

losing expert participation [Jud72]. Supporters respond primarily by trying to outline what is and is not a true Delphi study [RW99] citing poor implementation as incorrect use by the researcher. In addition new technologies such as the internet minimize “hurry and wait” by shortening the delivery time of next rounds to participants [Sku07].

Criticism of the method also focuses on the use and behavior of experts. Sackman argued that Delphi’s use of valid expert opinion is “scientifically untenable and overstated” [SZ75] and Delphi responses are likely to be ambitious [Sac74]. In addition, experts are not held accountable for their opinions even when their views are extreme [Sac74]. The use of a group of experts is motivated by the principle assumption that group decisions are better than those made by a single individual [MJH95a]. Delphi’s effectiveness in forecasting [Win97] gives support to that claim. Further, supporters argue that even in scenarios where extreme expert opinions are present, those with more extreme views are more likely to drop out of the study leading to consensus [Bar84].

Finally, criticism also exists of the process itself with critics arguing that Delphi falls outside mainstream methods of scientific questionnaire development [Sac74]. Supporters argues, however, that the Delphi method is no less methodically rigorous than other qualitative methods such as interviews [Tur70]. Ultimately, criticisms and support for Delphi reflect biases inherent with the process. Delphi’s anonymity can be seen as eliminating the adversary process [Sac74] or minimizing the effects of domineering experts [Mar91]. Delphi’s iterative examination can either be seen as discouraging exploratory thinking [Sac74] or reaching consensus [MJH95b]. It is the goal of the researcher to understand what this system affords and removes to identify whether or not the process is appropriate for their target goals. As there is longstanding support for the process’s effectiveness in educational planning [WS74], rubric creation through Delphi is well motivated. The section below highlights several of these educational uses as they pertain to computer science.

### **2.3.3 Applications of Delphi**

The Delphi process is widely used in education research and is beneficial for tackling hard problems where gathering people in the same place might be difficult, or when in-person situations or lack of anonymity might cause unwanted difficulties. In an educational application of the Delphi technique, Weaver (1971a) listed the following areas in which the technique could be used: (a) a method for studying the process of thinking about the future, (b) a pedagogical tool or teaching tool that forces people to think about the future in a more complex way than they ordinarily would, and (c) a planning tool that may aid in probing priorities held by members and constituencies of an organization (p. 271). For instance, Kloser used a Delphi study to identify practices core to teaching secondary level science classes [Klo14]. For his study, Kloser created a 3-round Delphi panel of 25

experts; 10 high school science teachers and 15 science education researchers from Tier 1 research universities. In order to ensure a wide variety of teachers and get varying viewpoints, high school teachers were selected from across the country with no more than two teachers working in the same state. For the university researchers, it was decided no more than two per university. Expert qualifications for both parties were determined by national or state honors, level of degree, and years of experience either teaching high school science, or teaching teachers [Klo14]. After the three rounds of voting on 10 starter practices and 50 nominated ones, the group came up with 9 teaching practices for high school teachers that they found to be very important. This also included coming up with a common language that would satisfy all of the experts, as discourse around certain terms was high.

One of the earliest uses of the Delphi method as it pertains to computer science is a 1985 study on the Emerging Computer Science Curricula. The mandate of the 1983 Nation at Risk report required that one half year of computer science be required of all high school students. The primary question at the time was: What is computer science? Gomez used a triangulation of grounded theory approach and the Delphi method to generate three main theories: Theory related to definition, theory related to curriculum, and theory related to support essentials. From these theories, a detailed description of computer science, a suggested curriculum, and recommendations for support systems were derived [Gom85]. It should be mentioned, that Gomez's work was in attention to Arizona specifically. Gomez started her research by selecting a panel of 9 experts and constituencies. These included university professors, professional educators, an aide to a public official, and members of a lay group (governing board members). She interviewed each member asking them a series of questions on how they see computer science, what supports might be needed for teachers, and what should be in the curriculum. From these interviews, she coded each statement into shorter keywords or phrases. As Glaser recommended, "if all data cannot be coded, the emerging theory does not fully fit and must be modified" [Gla78]. It was important to her to memo down ideas on how these codes connected as she was working. After processing the interviews and related literature, she generated a list of 108 statements pertaining to the three categories: definition, curriculum, and support. Using the same pool of experts who generated the data set used during coding, Gomez sent out the initial list of statements for experts to agree upon. Agreement in this case was on a 5-point scale (Strongly Disagree to Strongly Agree). The data were then sent back and forth between the panelists and the research team as is typical in most Delphi studies. She ended up performing three rounds of agreement. In each subsequent round she would send the specific panelist back their result in addition to the overall groups result. The panelist was asked to reassess their own answer, however, if the answer provided was outside of the interquartile range of the group, they would be requested to write a short statement supporting their selection. The outcomes of the study

were a list of recommendations related to both implementation and research of computer science education, in addition to a list of different computer science definitions for Business Education, Mathematics, and Science. Some of the key recommendations are in the computer science support section, such as: a course on the methods of teaching computer science, a skills requirement that includes becoming more knowledgeable about the subject matter and the mastery of different instructional techniques that are more appropriate to higher levels of instruction, and a certification requirement. As mentioned earlier these types of certifications and requirements are not as prevalent as one might hope. As of 2011, only South Carolina had a certification requirement for high school computer science teachers [Wil11].

A newer use of Delphi has been in the recommendations for competencies and learning objectives for a Game Art and Design class in North Carolina [Mac11]. In this four-round Internet-based study, the Delphi panel consisted of experts from the Independent Game Developers Association (IGDA) game education special interest group. Members were solicited and those interested responded to the accompanying demographics survey. The required criteria that determined which members were chosen as expert panel members included having at least two years of experience with gaming, and having at least two semesters experience teaching gaming. Thirty three members were selected, three for the review panel, and thirty for the expert panel. The expert panel was utilized for creating the list of learning objectives and the review panel was used to maintain consistency with the instruments and editing. The initial instrument sent out in round one was based on the nine Core Topics developed by IGDA. Panelists had space to add additional competencies or modify the wording of the ones provided. Once results were sent back in, the researcher made edits and sent the instrument to the review panel to approve. This helped eliminate bias that the researcher might have introduced. Round Two gave experts the option to Accept, Reject, Modify, or Add new competencies and rate them on a scale of 1 to 5, five being the highest. Competencies and objectives with a rating of 3.01 or higher were then moved on to round three. Round three involved ranking the remaining competencies for importance. These results allowed the researcher to correlate the ratings to the medians from previous rounds and enable the researcher to further observe favored competencies while identifying consensus if any. The final round was a simple Accept or Reject of the remaining competencies; no modifications were allowed at this stage.

Goldman et al. used three Delphi study panels of 20, 20, and 21 experts to generate core concepts for three computing intro courses: discrete math, programming fundamentals, and logic design [Gol08]. Each study was run in the same manner. In Phase 1, experts were to recommend 10 - 15 core concepts for the topic. These were collated by 2-3 independent researchers who first reconciled the list by themselves, then together. The summary of results was sent back to the experts for Phase 2 where panelists rated on scale of 1 to 10 the importance, difficulty, and expected mastery of each

suggested concept. Again the data were summarized and sent back; this time when panelists went to rate an item they would have to provide a justification if their vote was outside the interquartile range of the previous round. Meaning that if their vote was one deviation outside the group average, they would have to justify their strongly contrasting opinion. Finally, in Phase 4, expert panelists were given one last chance to rate the suggested concepts. Findings showed that over time the standard deviation dropped, as expected. A final list of the top 10 concepts for each class was created. Goldman et al. suggest that readers should take the concepts at face value, as experts agree these are important things that need to be focused on. However difficulty rating should be taken with a "grain of salt" as teachers might have different difficulty gauges and have incomplete understanding of student learning. Goldman suggests future interviews with students to get a wider view of concept difficulty. Experts might have a blind spot to concepts where students have frequent misconceptions.

## **2.4 Discussion on CS Education**

Based on the relevant literature we can see that computational thinking, and identifying ways to support teachers in the computer science classroom are critical to the success of CS Principles, and the adoption of computer science in general as a standard course offering in K-12 schools. In the push to get more students into K-12 computing classes, we not only need to develop and evaluate the courses, but also provide teacher training and support. Most states lack a computer science certification requirement, and many teachers come in with a minimal background in computing. From a CSTA meeting held at NC State's campus we had one teacher say, "I don't know what the students are supposed to be learning after teaching this unit (referring to a BJC lab); how am I supposed to grade it?" For these reasons, this research is conducted to help identify associated learning objectives using group decision techniques such as the Delphi, and translate them in such a way that novice CS Principles teachers can use them in their classroom.

## CHAPTER

# 3

# STUDY 1: SYSTEMATIC RUBRIC DEVELOPMENT FOR CS PRINCIPLES

*(Best Paper Nominee) Cateté, V., Snider, E., & Barnes, T. (2016, July). Developing a rubric for a creative cs principles lab. In Proceedings of the 2016 ACM Conference on Innovation and Technology in Computer Science Education (pp. 290-295). ACM.*

### **Abstract**

The “Beauty and Joy of Computing” Computer Science Principles class has inspired many new teachers to learn to teach creative computing classes in high schools. However, new computer science teachers feel under-prepared to grade open-ended programming assignments and support their students’ successful learning. Rubrics have widely been used to help teaching assistants grade programs, and are a promising way to support new teachers to learn how to grade BJC programs. In this paper, we adapt general coding criteria from auto-graders to a lab where students write code to draw a brick wall. We tested the rubric on student assignments and showed that we can achieve high inter-rater agreement with the refined rubric.

### 3.1 Introduction

Beauty and Joy of Computing (BJC) is a version of Computer Science Principles (CSP), a new Advanced Placement course, created as an attractive and engaging introduction to computer science that can be taken in high school for college credit. BJC is a rigorous course that uses open-ended *Snap!* programming problems to teach the seven big ideas of CSP: creativity, abstraction, data and information, algorithms, programming, the internet, and global impact. With funding from the National Science Foundation, the BJC project has provided professional development for over 140 high school teachers nationwide. Teachers enjoy the course and it is well-received by students from diverse backgrounds [Pri15], but teachers report that it is difficult to assess student work and provide students the detailed feedback they need to grow in their computing skills. Most BJC teachers do not have a computer science background, and therefore have no experience evaluating code, or even in understanding how the complex, open-ended programming labs relate to the seven big ideas of the course. Therefore, it is critical that we help new teachers understand how the tasks students undertake in their labs can be assessed, and how their performance relates to the course's learning objectives.

To address this need, we are defining rubrics for BJC labs that teachers can use to grade student programs. We envision that these rubrics will be usable by new teachers for assessment, and for providing students with meaningful feedback that links to the course's important content. This paper represents the first step in our process - identifying common novice programming grading criteria, adapting them to a specific BJC lab, testing their application on student programs, and refining the rubric to obtain good inter-rater reliability. While grading student programs is straightforward for experienced computer scientists, inexperience with computing and the variety of student code makes this a difficult task for teachers. Less experienced graders must rely on the stated assignment objectives, which can be difficult to translate into numeric grades.

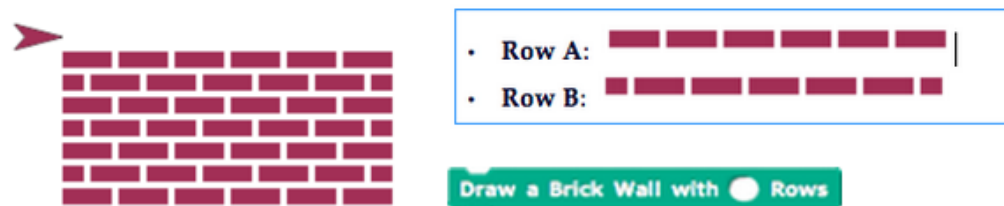
We used a three-phased approach to design and evaluate a rubric for the Brick Wall assignment. In Phase I, we first selected rubric criteria based on literature on automatic support for grading novice programming assignments, and wrote descriptions for 4 levels of performance for each criteria to create an initial rubric. Then, two raters applied the initial rubric to grade 9 student programs independently, and we measured inter-rater reliability. In Phase II, the raters compared their assessments, and refined the rubric and assessments of the initial 9 samples until they agreed upon the scores for each of these projects. Finally, the raters tested the robustness of the revised rubric on 10 additional college assignments, and computed Kappa to measure inter-rater reliability on the new assignments. We considered a Kappa value of 0.7 or higher to be reliable. In Phase III, we rated all of the projects to compare the results for students taught by an expert high school computer

science (CS) teacher with those taught by novice undergraduates.

This work is important in demonstrating that it is possible to create a rubric based on computer science education literature, that can be reliably applied by people with no computing background, to programs that are actually created by novice programmers. Our results show that we can achieve high inter-rater reliability between a computer scientist and a science educator with no computing background, on diverse programs created in labs with teaching assistants with no prior programming experience. These results provide strong evidence that we can adapt a general rubric to help novice computing teachers evaluate student programs for particular BJC labs. In the remainder of Section 1, we present the Brick Wall assignment. In Section 2, we explain the development of the Brick Wall rubric. In Section 3, we describe the study methods and phases. In Sections 4 and 5, we present our results and conclusions.

### 3.1.1 Brick Wall Assignment

Surveys show that most high school BJC teachers from 2012-2014 used Brick Wall, so we chose it for rubric development. BJC's Brick Wall lab was designed to demonstrate abstraction and the value of creating a function that can be called to perform the same task multiple times, and with different parameters. The main objective of the Brick Wall lab is to create a brick wall with an alternating pattern of bricks. In the assignment, two row types A and B are defined, where an A row is made up of whole bricks, and a B row starts and ends with half-bricks. The assignment specifies that students should create a new block in *Snap!* that takes 'number of rows' as input and draws a brick wall that alternates A and B type rows accordingly, as shown in Figure 3.1. Students are instructed to create one method that generates a brick wall, two separate methods that draw the two row types, and a method that draws an individual brick. These specific instructions are explained as levels of abstraction for solving the problem. The visual nature of the task and the clear and simple repetitive structure of a brick wall are affordances that should help make iteration and functions seem to be natural solutions to the problem.



**Figure 3.1** Levels of abstraction for the Brick Wall assignment

## 3.2 Rubric Creation

To support new teachers to teach computer science, we would ideally build automated systems that could provide both students and teachers with feedback on student performance. Several such intelligent tutoring systems (ITS) have been used to teach programming to novice students. However, the new BJC course strives to inspire new and diverse students to explore computing with modern tools that can be used to express creativity, such as the *Snap!* programming language. To more quickly support teachers with these new environments, we determined it would be helpful to develop rubrics to help teachers understand the goals of each programming assignment. Our rubric development was strongly influenced by ITSs for programming, since we wish to make an objective system that can be used reliably by teachers with different backgrounds to accurately grade student work. In this section, we describe our process to create a rubric based on computer science education literature. We draw strongly from the literature in intelligent tutoring, where researchers have expended great effort to automate the processes needed to evaluate student code.

### 3.2.1 Rubrics in CS Ed

In education, rubrics are used as performance-based assessment for student projects. Rubrics use descriptive measures to separate levels of performance on a given task [SL05]. Then, each level of performance for a particular task is assigned a value. Values in rating scales on rubrics can be either holistic or analytic [Bro13]. Holistic rubrics are quick and efficient, evaluating projects as a whole. Analytic rubrics take longer to design, but offer multiple dimensions that are evaluated separately. Since our goal was to enable teachers to provide more detailed feedback, we chose to design an analytic rubric.

Rubrics can effectively support teaching assistants, who are non-experts in computer science, to quickly, accurately, and consistently assess novice student programs for style, functionality, and design[Bec03]. Researchers tend to agree on three levels of achievement ratings that denote shortcomings, meets expected outcomes, and goes beyond expectations [Bec03; Fit13; Ste14]. Stegeman separates expected outcomes into two levels - almost there, and meets expectations [Ste14], and we use this refinement to arrive at four levels. While there is agreement on the levels that programming rubrics should contain, McCauley highlights that it is important that rubrics have precise and consistent language [McC03]. Toward that end, we chose our rubric criteria from those developed for automated support for grading, as described in the next section.

### 3.2.2 Criteria Selection

The first step in designing a rubric is to identify a coherent set of criteria for evaluation [Bro13]. When designing our rubric to help teachers evaluate student projects by hand, we wanted to focus on evidence that would be clearly observable in student programs. Since automated systems rely upon objective and observable criteria, they are a natural source for our rubric criteria: Accuracy (correctness), Efficiency (as measured by the number of times blocks are repeated in code), Reasoning (domain knowledge), and Readability (style). These criteria were inspired by Pillay, Zeller, Jackson and Usher as we discuss here.

In 2003, Pillay studied four separate programming tutors and identified four common ways they evaluate student code [Pil03], including: accuracy or correctness of topic, efficiency of the code, adherence to style guidelines, and algorithmic domain knowledge. Domain knowledge is comprised of procedural knowledge on how to write programs and declarative knowledge on how to apply different programming concepts. Since these were common features, we included all four in our initial rubric design.

Jackson and Usher's ASSYST auto-grading system was built to alleviate the errors that people are prone to make when grading, while still keeping a human tutor involved [JU97]. ASSYST assesses student code for: correctness, efficiency, style, complexity, and test data adequacy. ASSYST measures correctness by how well the student program's output matches the output from a known correct program. Correctness, in BJC, corresponds to whether the program achieves the stated goal, and corresponds to the accuracy metric in Pillay's work and our rubric. ASSYST measures efficiency by keeping track of how often a particular block of code is executed. This assigns higher efficiency to a loop which runs a single command 10 times than code for the same task that uses 10 separate commands each executed once [Hic13]. When scoring code by hand, we estimate this by observing whether the projects contain identical code that is written more than once. Repetitive commands in code represent less efficient code, where a loop could have been used instead of copy-paste. ASSYST's style metric includes features such as indentation and variable naming. A similar metric can be found in Pillay's work, as well as in Zeller's readability metric, that refers to how easily another person can read and understand how the project code achieves the goals [Pil03; Zel00]. We combined these to produce a single Readability criterion for our rubric.

### 3.2.3 Performance Descriptions

Once we identified the rubric criteria (Accuracy, Efficiency, Reasoning, and Readability), we used the Brick Wall assignment to write descriptions of observable behaviors for each level of performance for each criteria. Brookhart's guidelines suggest that performance level descriptions should be clear,

descriptive, and cover the whole range of performance [Bro13]. We started by identifying Level 3 for meets expectations, as described in the assignment. Level 1 descriptions were basically the opposite of these. We developed Level 2 descriptions by identifying how students might have just missed meeting expectations. To develop Level 4 descriptions, we added language about how students might enhance their programs along the lines of the given criteria. Table 3.1 lists our rubric, with initial descriptions in plain text and the refinements made to create the final rubric added in italics.

**Table 3.1** Final rubric used for initial grading of student projects in Brick Wall assignment. NOTE: item in italics are additions made to the rubric for clarity.

Category	4	3	2	1
Accuracy of Content	At least 5-6 rows of bricks created <i>with alternating brick pattern but all rows same length</i>	Two rows of brick created with alternating patterns, <i>or 5-6 inaccurate rows</i>	1 brick created and repeated	1 brick created
Efficiency	Abstracted methods and loops used to repeat coding patterns	A few abstracted methods were created, but there is still a lot of repeated code	Code works but is repeated simple statements ( <i>little abstraction</i> )	Abstraction of complex methods not identifiable
Rules and Reasoning	There is evidence of the use of mathematical and logical concepts & appropriate use of abstractions and algorithms.	There is evidence of the use of mathematical and logical concepts <b>or</b> appropriate use of abstractions and algorithms.	There is inappropriate use of abstractions and algorithms.	Little or no evidence of the use of mathematical and logical concepts exists
Readability	The source code has been commented and the source code is correct, logical, & easily readable.	The source code is correct, logical, and easily readable.	The source code is mostly correct, logical, and readable.	The source code is unclear, incorrect, or incomplete.

### **3.3 Study Design and Methods**

Our hypothesis was that we could develop a well-defined, analytic rubric, that raters can use to achieve an acceptable level of inter-rater reliability (with Kappa at least 0.7). The following sections describe the context, data selection, and methods for rating and rubric refinement.

#### **3.3.1 Context and Data Sources**

Student code samples for rating were drawn randomly from three classes: two high school elective BJC classes taught by the same teacher in Fall 2013, and one college introductory non-majors computing course taught in Spring 2014. The high school data was from one high school teacher teaching two sections of BJC in Fall 2014. Of the original 42 students, one joined the course halfway through and 2 were missing files, leaving a total of 39 high school submissions. In the Spring 2014 college course with 78 students, 57 files were submitted but 2 were unusable, leaving 55 college submissions. A total of nineteen projects were randomly chosen for analysis, with nine (three from each course) for the initial round, and 10 from the college course for the final round. These projects ranged from completely accurate walls, to one brick telling a joke to a circle.

The two high school classes were taught by an experienced AP Computer Science high school teacher. The college course had lectures by a Computer Science professor, but the seven lab sections were facilitated by five undergraduate teaching assistants (UTAs) with no prior experience in teaching or in BJC. This is similar to the preparation that new BJC high school teachers have, ranging from inexperienced teachers with no computing background to an expert with 20 years of experience teaching advanced computer science in high schools. The five UTA majors were: computer science, civil engineering, materials science, paper science, and textile technology. The UTAs had no prior teaching experience, and no similar computing course. The course professor led the UTAs through the lab the week before students did it, in much the same way that high school teachers keep one step ahead of their students when they teach a new class for the first time. We therefore felt that code from the college course would be representative of work we might expect from students taught by an inexperienced high school teacher with no computing background. Therefore, random samples were chosen from the college course for the evaluation of the final rubric.

#### **3.3.2 Methods**

In each phase, two raters worked independently using the rubric to grade each submission, and we computed Kappa for inter-rater reliability at the end of each phase. Rater1 has a Master's in Computer Science, and Rater2 has a PhD in Science Education. In Phase I, both raters rated nine

samples from the three courses (three samples from each). To make the selections, we skimmed the programs and determined that the high school submissions were similar, so we selected three from each high school course at random. Since we wished to have programs demonstrating diverse performance levels, we made random selections from the college course but continued to make selections until we felt we had samples that were fair, good, and very good.

In Phase II, the raters compared their ratings of all nine samples, and when they did not agree, discussed the ratings, and refined the rubric, as shown in italics in Table 3.1, until they achieved 100 percent agreement on the ratings for the 9 samples. We will elaborate on some of the contentious coding samples in section 5.2. During this comparison, the raters realized that the quality of submissions from the high school course was higher and less variable. Therefore, the raters selected 10 projects from the college course to test the final, refined rubric for Phase II. As in Phase I, the college projects were selected randomly but samples were excluded if they were very similar to previously selected samples.

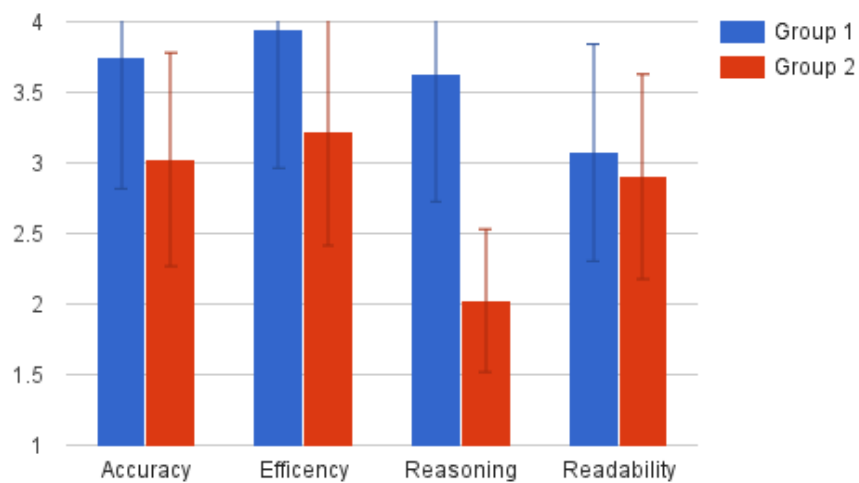
Finally, in Phase III, we trained a third-year undergraduate computer science major to apply the final rubric to the Brick Wall assignment. Rater1 and the new undergraduate (UG) Rater3 applied the final rubric to all 94 Brick Wall assignments, and computed average composite scores for the high school and college projects to compare the performance between the classes. We hypothesized that the experienced high school teacher's students would outperform college students facilitated by university teaching assistants who were inexperienced at both teaching and CS.

### **3.4 Results**

In Phase I, three projects from each section (6 high school, 3 college) were rated using the initial rubric. We found very low agreement between our raters, with a Cohen's Kappa of 0.37. In Phase II, the raters discussed their ratings and refined the rubric, clarifying the descriptions, until they agreed on all ratings for all criteria on the 9 samples. The final rubric is shown in Table 3.1 with changes between initial and final in italics. In Phase II, the raters applied the resulting final rubric to rate 10 additional college coding samples, and achieved a high level of inter-rater agreement, with Kappa of 0.73. In Phase III, the newly-trained Rater3 achieved a Kappa of 0.735 with Rater1 and Rater 2 on the 19 projects rated in Phases I and II. Rater 1 and Rater3 then rated the remaining 75 projects and aggregated the rubric scores by high school and college. Results show a clear distinction in performance in labs between the students taught by novices versus the expert high school computer science teacher.

High school students taught by the "master" teacher (group 1) averaged 3.60 on the Brick Wall assignment. College students taught by novice undergraduates (group 2) scored an average of 2.79

on the assignment. Figure 3.2 shows the criteria scores for each class with 25 percent error bars, demonstrating that the high school students taught by a master teacher scored quite highly on accuracy and efficiency, and similarly to the college students taught by novices in readability. The largest difference occurs in the reasoning category, where high school students were more likely to use conditionals and mathematical expressions. This difference could be due to the teachers, but could also be due to a difference between the self-selection of high school students into a pre-college course, and students taking a low-level computing course as a general analytic elective in college. We present these results to show how the potential differences between novice and master BJC teachers may be impacting student performance in BJC courses.



**Figure 3.2** Rubric scores for group 1 and group 2

To illustrate the application of the rubric to student code, we present sample projects that demonstrate low, medium, or high levels of overall performance, based on standard deviation (SD) away from the overall average for the 19 scored projects. Students more than one SD below average were ranked Low, students one SD above were ranked high. The remaining programs were ranked medium. Of the six high school projects, one was ranked Medium and five were ranked High for

demonstrating appropriate abstraction, use of parameters, programming logic and mathematical reasoning. The 13 college class artifacts were rated with two High, seven Medium, and four Low.

### 3.4.1 Code Examples

In this section, we showcase one example at each performance level (low, medium, and high). Figure 3.3 shows an example of a low project with the average of all four ratings of 1.75. The Accuracy score was 3 for creating two rows of bricks with alternating lengths. The Efficiency score was 2, since the code works but consists of repeated simple statements. The Reasoning score was 1, as there is little or no evidence of the use of mathematical and logical concepts exists. Finally, the Readability score was 1, as the source code is unclear, incorrect, or incomplete.

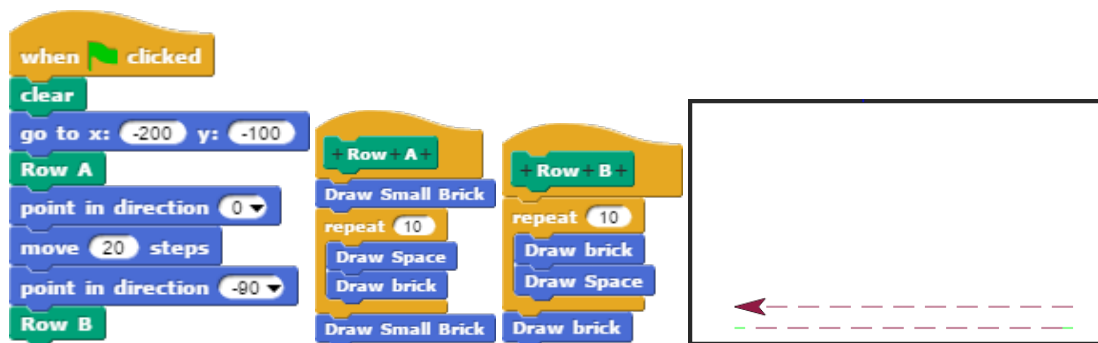
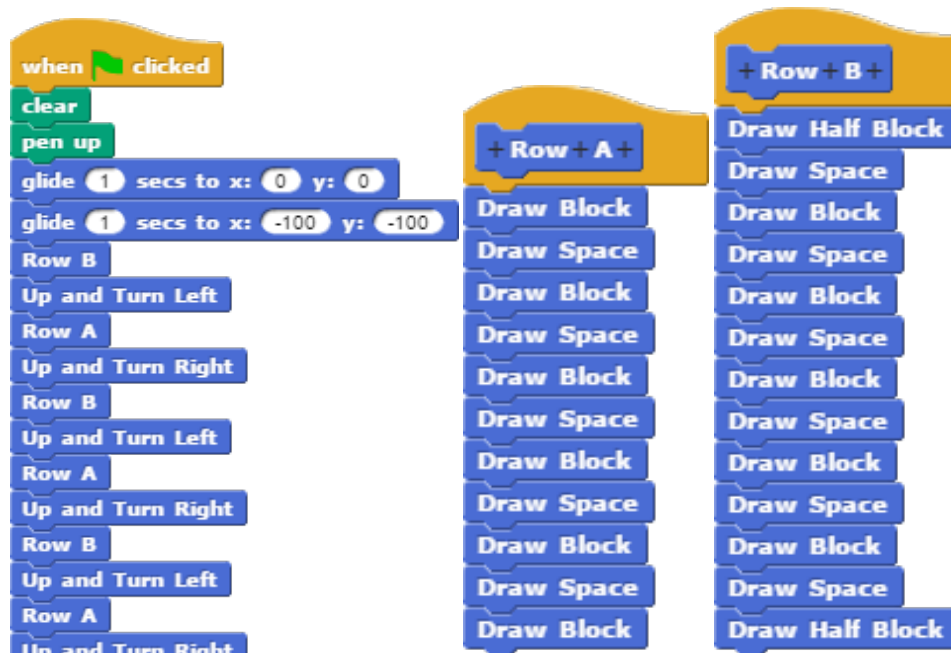


Figure 3.3 Low-level example, scored as: Accuracy = 3, Efficiency = 2, Reasoning = 1, and Readability = 1.

Figure 3.4 demonstrates medium performance, with an average score of 3.0. The Accuracy score was 4 for having at least 5-6 rows of bricks with alternating lengths. The Efficiency score was 3, since a few methods were created, but there is still a lot of repeated code. The Reasoning score was 2 since there is inappropriate use of abstractions and algorithms. Lastly, the Readability scored 3 since the code is correct, logical, and easily readable.

Figure 3.5 demonstrates high overall performance, with an average score of 3.75. The Accuracy score was 4 since the program creates at least 5-6 rows of bricks with alternating lengths. The Efficiency score was 4 for using methods and loops to repeat coding patterns. Reasoning also scored 4 since there is evidence of the use of mathematical and logical concepts and appropriate use of abstractions and algorithms. Finally, the Readability scores 3 since the source code is correct, logical, and easily readable.

These three student code examples demonstrate the diverse ways students can complete an



**Figure 3.4** Medium-level example, scored as: Accuracy = 4, Efficiency = 3, Reasoning = 2, and Readability = 3. This program shows lack of loops and logic

assignment. The low example uses a simple repeat loop, but only creates two rows. The medium example, forgoes loops, rather listing each row separately. The medium and high examples both draw a complete brick wall, but only the high example demonstrates how abstraction and parameters can be used to draw a brick wall customized to user input.

### 3.4.2 Rubric Refinement

There were four main areas where we edited the initial rubric to resolve rating differences, three in accuracy: misaligned row ends, parameter setting, and code needing edits to run properly; and one in reasoning, with student use of “magic numbers”. The Accuracy criterion relies on matching the program’s output to the lab assignment’s objectives. The lab assignment was generally detailed enough for the two raters to consistently score projects for Accuracy, but a few student programs demonstrated behaviors not accounted for on the rubric. Readability measures how well another person can read someone’s code. This category was easily measurable by both raters and adjustments were not required. Efficiency is related to code length - shorter code with blocks that are executed more often than they appear in the program is more efficient. This difference can be seen between Figure 3.4 which uses the ‘Up and Turn Left’ and ‘Up and Turn Right’ blocks each five times, versus

```

+draw a brick wall with number of rows rows and
numberofbricks bricks and a brick size of bricksize and
length of lengthofbrick and mortar size of mortarsize +
if number of rows mod 2 = 0
repeat number of rows / 2
row A with numberofbricks bricks and brick size of bricksize and
length of lengthofbrick and mortar size of mortarsize
transition between row A and B with mortarsize space and brick size
bricksize and number of bricks numberofbricks and length of bricks
lengthofbrick and mortar size mortarsize
row B with numberofbricks bricks and brick size of bricksize and
length of lengthofbrick and mortar size of mortarsize
transition between row B and A with mortarsize space and brick size
bricksize and number of bricks numberofbricks and length of bricks
lengthofbrick and mortar size mortarsize

```

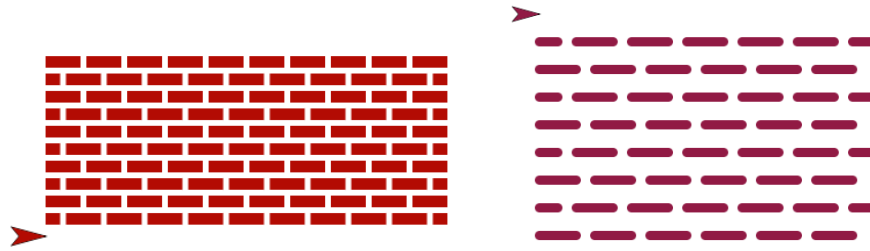
(a) First half of method to generate wall (even # of rows)

**Figure 3.5** High-level example, scored as: Accuracy = 4, Efficiency = 4, Reasoning = 4, and Readability = 3. Program uses loops & math to generate correct output.

Figure 3.5 which uses each transition block once (it gets repetitively called using a loop). Since this was a straightforward measure, the raters agreed on scores. The Reasoning criterion describes use of logic and abstraction - such as having if or while blocks. The main difference in raters on this was that Rater1 could determine whether mathematical expressions logically matched the program requirements, or if a student inserted a parameter, we call a “magic number” to adjust program output to be correct, despite this number having no basis in logic.

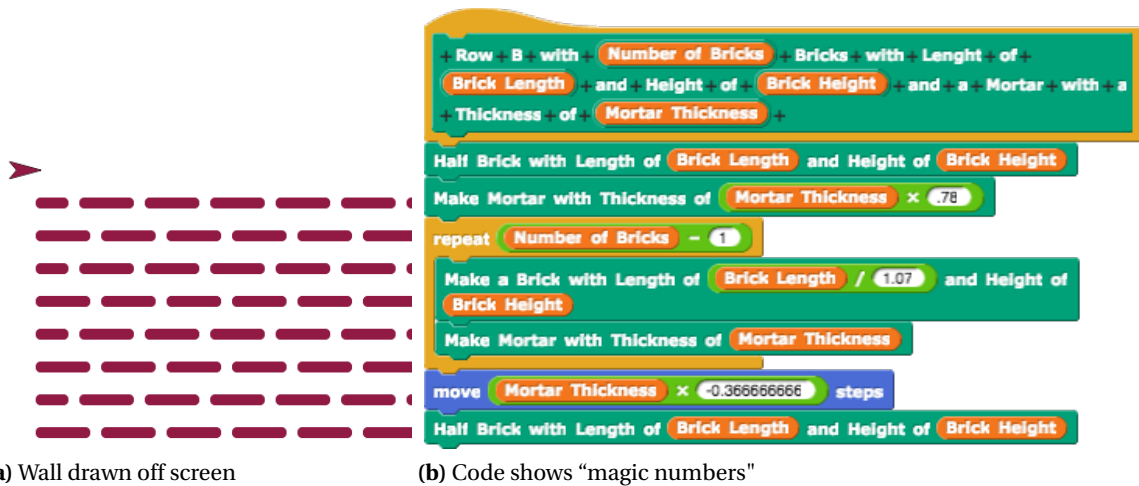
The primary confusion between raters typically revolved around different interpretations of Accuracy. The first difference was in how to score Accuracy for code that correctly draws multiple rows of bricks but whose ends did not align, as shown in Figure 3.6. Originally, the rubric listed 5-6 rows as a 4 rating and 2 rows as a 3 rating, since we anticipated that some projects would not include loops to create multiple rows. Rater1 scored rows with alternating lengths with a 3, since they were not entirely accurate, while Rater2 scored them with a 4 since they had 5-6 rows. Therefore, we added the words “or 5-6 inaccurate rows” to the Accuracy level 3 description to settle this difference.

A second difference between the raters was in judging Accuracy for programs that had adjustable



**Figure 3.6** Two brick walls with differing row alignment on the right side.

parameters and inputs. In some student projects, the brick wall would not be drawn exactly right unless the rater modified a parameter or input before running the code. Rater1 (computer scientist) scored these higher than Rater2, since it is standard practice in computer science courses to test code with multiple input values. Rater2 was not familiar with this practice, so rated these projects as incorrect. Upon discussion, the raters agreed to count programs as correct if parameters could be easily adjusted to achieve the correct output, but this was not explicitly added to the rubric.



**Figure 3.7** Blocks showing student misunderstandings.

The raters discovered that some adjustment of the *Snap!* interface might be needed to fully observe some program behaviors. For a few projects, zooming out or re-centering the stage view made it possible to view the entire wall for evaluation. For a few others, the student had programmed

fixed off-screen values for the start and end positions of the wall, relying on the *Snap!* interface to cut off the wall edges (see Figure 3.7 a). This did not require adjustment of the rubric, but rather adjustment in how the raters used *Snap!* to run programs.

The use of “magic numbers” was the source of confusion for both raters. It was apparent in one example (Figure 3.7 b) that a student was having difficulty aligning the A and B-type rows and relied upon formulas or numbers to adjust the placement of bricks. However, these “magic numbers” were not applied systematically or in a way that appeared to have a logical reason (to either rater). In these cases, the brick walls looked good but it appeared that students applied a guess and check strategy to determine numbers to use to make the brick wall look correct. The intention of the assignment was for the students to discover a general solution to the brick wall problem, not one that is specific to a particular browser, screen resolution, or idiosyncrasies of student code. In this case, math and logic are clearly applied (since there are mathematical operators visible in the code), but, without a clear formula, it was unclear as to whether or not math and logic were being applied appropriately. This project was also one that had pre-programmed positions for the starting point, and attempts to center the brick wall on stage were difficult to make. It was resolved that there was in fact (inappropriate) use of magic numbers, and another field was added to Reasoning level 2 (Table 3.1).

### **3.5 Conclusions and Future Work**

In this study, we created an analytic rubric based on evaluation frameworks that automatically assist people in grading student programs, applied the rubric to a set of student work, refined the rubric, and applied it again, demonstrating a dramatic increase in inter-rater reliability. Our rubric rates BJC Brick Wall programs for accuracy, efficiency, reasoning, and readability. In Phase I, we observed that Rater2 (science education) initially had more difficulties with the rubric. In Phase II, after discussion and strategic updates to the rubric, this rater’s scores aligned much more closely with those by Rater1 (computer scientist). One limitation of the study is that the raters’ discussion likely informed the scoring of assignments in Phase II, so therefore do not represent the likely scores of novice teachers upon first application of the rubric. However, to help address this limitation, we have selected sample student programs that illustrates three levels of proficiency, and will provide these types of samples in our future studies of rubrics as they are used by novice BJC teachers. We also believe that teachers will learn, from the application of the rubric to student code, to differentiate programs according to the rubric criteria, promoting their learning of computer science and their skill in evaluating student programs. Eventually, we believe teachers can use the rubrics to guide students to create better programs.

Another limitation of this study was that the Brick Wall lab is relatively simple. This may be the reason that only minor changes were needed to achieve an acceptable rubric. For more complex assignments, it may not be as straightforward to map the assignment to the rubric criteria. We expect that the readability and efficiency criteria will not need much refinement, but accuracy criteria may need to be added for each program objective, and reasoning may also need individual criteria for each distinct objective.

In future work, we plan to align our rubric with essential knowledge components from the CS Principles course framework available at [apcsprinciples.org](http://apcsprinciples.org). For example, Essential Knowledge component 4.1.1 D enumerates the learning goals for Mathematics; specifying iteration, loops, and using the modulus operator. Raters considered similar ideas within the Reasoning category of our rubric, but we could revise the rubric to be more focused on learning objectives, and this is better for providing students target learning goals [Bro13]. By expanding the Reasoning section of our Brick Wall rubric to focus on learning goals and not just task completion, teachers will also be able to better identify what students should be learning and doing during a particular activity or lab.

## CHAPTER

# 4

## STUDY 2: TASK VS. LEARNING BASED RUBRIC EVALUATION

### 4.1 Introduction

Today Computer Science plays a critical role in nearly every field. There is a growing body of literature that recognizes the importance of improving K-12 computer science education and including computational thinking in earlier grades and other subjects [BS11; Tuc03; Win06]. One of the biggest roadblocks to increasing the pervasiveness of computing classes is the limited number of well qualified computing teachers available to teach the courses. The CS10K movement is pushing to create 10,000 new computer science teachers; many of these teachers are converting from other disciplines like math or business. A primary concern for K-12 CS education researchers is the lack of preparedness of our new computer science teachers [Ben15; Cun14; Goo14]. These new HS CS teachers have experience in the K-12 classroom teaching other subjects and understanding pedagogical theories. Unfortunately, these teachers do not fully understand the concepts behind computer science and computational thinking; key elements in the new AP Computer Science Principles (CS Principles) course. As CS Principles is a curriculum framework that comes in multiple forms, we will narrow our research by focusing on the development of quality support materials for

the Beauty and Joy of Computing (BJC) implementation of CS Principles.

Through intensive professional development workshops and in-person interviews, we have found that many BJC teachers are struggling with assessing student work in the form of programming lab assignments. These student solutions are written in the Snap! programming language and the labs are intentionally open-ended to allow for student creativity. Whereas traditional computer science professors and teaching assistants can recognize computing algorithms and concepts in students' code, the BJC teachers have not had nearly as many years in training (most teachers have only had one or two computing classes while in college). Additionally, as the teachers were handed a pre-built curriculum, they are not confident in their understanding of what the particular learning goals are for individual labs or how to evaluate them. In our research, we explore the value and efficiency of developing performance-based versus learning-based rubrics for the support of and use by novice computer science teachers in evaluating BJC programming labs as measured by rater-reliability, ease of use, and student score distributions.

## 4.2 Background

In K-12 Computer Science Education, it is understood that assessment is needed to evaluate student learning and computational thinking abilities. However, there is still no general method on how to develop these assessments reliably and easily. The literature describes several techniques to create assessments for specific programming languages or development environments [Dan12; Fra13; Tay14; TG10], but these fail to support assessment evaluations by teachers who are not experts in computational thinking.

In Koh's 2010 paper, they use a computational thinking spiral developed for the Scalable Game Design (SGD) course to analyze what computational thinking patterns are in student code [Koh10]. When analyzing student projects (written using AgentSheets) the SGD researchers look for program behavior similarity, and also apply latent semantic analysis (LSA) to evaluate student code. The SGD research team created a Computational Thinking Pattern (CTP) Graph (a star plot with CT's listed as the outer vertices) to describe the types of learning evidenced in student code

However, as Koh states, the CTP graph content and structure are somewhat "arbitrary" – because the CS ed research community has not yet created a set of agreed-upon measures for computational thinking.. We aim to produce a set of computational thinking learning objectives, based on the CSP course, through our research. We use a Delphi method, described in Section 4.3, to generate learning outcomes and code samples for a particular CS Principles lab. A Delphi study is a technique to obtain the most reliable consensus of opinion of a group of experts using intensive questionnaires interspersed with controlled opinion feedback [DH63]. The outcome of this research will be a set

of rubrics whose content is agreed upon by experts, but whose structure is easy to use for novice computing teachers.

Rubrics are one of the most common performance-based assessment tools used to evaluate student projects. Rubrics use descriptive measures to separate levels of performance on a given task [SL05]. Then, each level of performance for a particular task is assigned a value. Rating scales on rubrics can be either holistic or analytical [Bro13; Mos03]. Holistic rubrics are quick and efficient; they rate the project as a whole and do not provide specific feedback on what elements of the project could be improved. However, these may not be effective in conveying to students how they can improve, and experts may in fact disagree on these ratings. Analytical rubrics take longer to design, but offer multiple dimensions for which students can get feedback. When using an analytical rubric each dimension gets treated separately to avoid bias from the end product.

Rubrics are important to use because each student project is rated individually against the same criteria. This avoids cases where projects are subject to a grader's own judgment, which may differ greatly between graders, instructors, or more specifically BJC teachers [Bos02]. Rubrics help clarify expectations for students while also speeding up grading. Instructors can also refine their teaching by evaluating rubric results to see which areas need improvement.

The creation of concept inventories and language specific guidelines are too strict for an open ended course like CS Principles. Alternatively, I have demonstrated the creation of a performance-based rubric for a CS Principles lab using grading schemes from the field of intelligent tutoring systems and program auto-graders [Cat16], which applies more directly to the CS Principles course. Although the performance-based rubric was used successfully, learning-based rubrics have been shown to be more objective [JS07]. The main difference in the two rubric types is that the learning-based rubric is learning goal oriented, while performance-based rubrics evaluate the end product.

In the following sections, we will describe the study methods for generating the improved learning-based rubric as well as the results of its application to student code samples when compared with a performance-based rubric. We conclude with discussion of the results and further work to continue to improve the support available to CS Principles/BJC teachers.

### **4.3 Summer 2015 Delphi**

A traditional Delphi panel calls for 15-24 experts on a subject to generate a consensus on a particular problem or task. The Delphi method elicits data from expert panelists through 3-5 rounds of surveys for use in creation of new expertly agreed upon materials. In the summer of 2015, we conducted a Delphi study on a CS Principles lab, Hangman, in order to create learning goal data for an expert generated rubric.

CS Principles Master Teachers were undergoing professional development for the Beauty and Joy of Computing (BJC) for one week in summer of 2015. These Master Teachers have all undergone previous BJC professional development, and were now being trained in advanced pedagogical techniques and training guidelines for the course. Due to their mixed backgrounds and recent training in CS Principles/BJC they made for ideal panelists in the Delphi study.

As the Master Teachers were considered BJC content experts and had recently reviewed the full curriculum as part of their master training, their opinions on applicable learning objectives and student code would help create objectives for a learning oriented rubric and give a perspective on expected student [artifact] outcomes. It was hypothesized that the panelists would adequately align learning objectives to a particular lab; however, their student code samples would differ from authentic student code, highlighting a gap in computer science programming knowledge and pedagogical experience.

#### **4.3.1 The Panelists**

A total of 9 Master Teachers participated as expert panelists for the Delphi study: six female, three male. Each participant was a current high school teacher with 1-3 years of past experience teaching CS Principles. 15% of teachers had 1 year of experience, 57% of teachers had 1-2 years of experience 28% had more than 2 years of experience teaching CS Principles. The primary courses taught by the panelists were: business management, mathematics, physical science, and computing. Two of the panelists have advanced degrees in teaching, two have Masters degrees in their subject field, and one of the teachers is National Board certified.

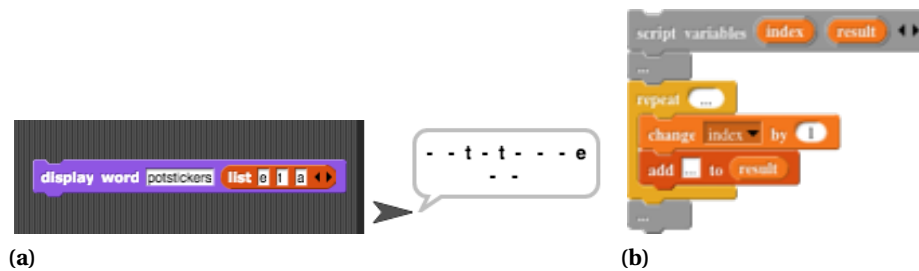
In order to complete the Delphi study, panelists answered a series of three moderately short surveys, which were based on the results of the previous survey. The details of each round of surveys are explained in Section 4.3.3.

#### **4.3.2 Hangman Lab Description**

The Hangman Lab is part of the Lists and Higher Order Functions units in the BJC curriculum [Uni14]. The lab was designed to combine tasks learned in exploring list operations such as ‘map’, ‘keep’, and ‘combine.’ This activity is more advanced than the Brick Wall lab and requires a better understanding of algorithms. The students start with code they used in creating a word list, with the description shown in Figure 4.1b. The directions for completing the Hangman activity are in the text that follows.

Imagine that you’re writing a program to play Hangman. The program has thought of a secret word, and the user is trying to guess it. Write a display word block that takes

two inputs, the secret word and a list of the letters guessed by the user so far. It should say the letters of the secret word, spaced out, with underscore characters replacing the letters not yet guessed: (Use your word->list block on the secret word to get started.) See Figure 4.1a below for an example.



**Figure 4.1** (a) New block to display a secret word ‘Hangman’ style. (b) Beginning structure of a ‘Word-list’ block.

Write a word->list block that takes a word of text as input, and reports a list in which each item is a single letter from the word. To do this, you’ll have to use a loop, along with the add to list block (above).

### 4.3.3 Survey Rounds

Our Delphi study contained four different rounds and focused on the BJC lesson Hangman. We selected Hangman for this study as it is one of the most widely used activities in the BJC community and features a more complex programming requirement than the Brick Wall Lab used in Chapter 3 [Cat16]. In the first round of this Delphi study, we asked participants to select all learning objectives from the AP CS Principles framework [Col14] that they believe apply to the Hangman lesson. We also asked participants to indicate the top five exemplars out of the objectives that they chose. A total of 22 learning objectives were chosen overall as relating to the Hangman lesson. Out of these, 17 learning objectives were selected as Top 5 by the participants. The top 18 learning objectives (80%) with the highest frequency of selection were chosen for inclusion for rating in the second round of the Delphi study. In the second round, the participants rated each of the included learning objectives on a scale of 1 to 5 (Strongly Disagree to Strongly Agree) on how much they agreed that the learning objectives related to the Hangman lesson. For learning objectives that were more general,

we also included the Essential Knowledge (EKs) indicators for participants to select. For example, learning objective 5.3.1 'Programming is facilitated by appropriate abstractions' was broken down into its more explicit EKs such as 'Integers and floating-point numbers are used in programs without requiring understanding of how they are implemented.' This allowed us to determine which aspects of the learning objectives participants felt most strongly about. The five learning objectives and six EKs with a rating of 4.25 or higher were selected to continue to the third round.

The third round of this Delphi study required the participants to generate code samples correlating to the learning objectives remaining from previous rounds. For each learning objective of their choice, participants wrote pieces of programs that highlighted that concept at high, medium, and low levels of understanding. All code samples were written in Snap, the programming language promoted by the BJC course materials. Each teacher submitted two code samples for a total of 18; however, two were removed due to being corrupted files. The primary learning objectives codified by our panelists were abstraction and using mathematical reasoning.

Given the three sets of high, medium, and low level programming samples and paired learning objectives, our team developed a new, learning outcomes based rubric (4.1) to use with the Hangman assignment. We discuss this rubric and its application by non-experts in the following sections.

#### **4.4 Rubric Evaluation Methods**

The learning-based rubric for Hangman references a total of six essential knowledge components and three learning objectives broken down into five value measures. The rubric uses expected learning outcomes (essential knowledge and learning objectives) as descriptive categories for grading. These categories and outcomes were based on feedback results from the Summer 2015 mini-Delphi Study. A detailed look at the rubric can be seen in Table 4.1.

In order to evaluate the effectiveness of the rubric in supporting consistent and meaningful grading results, two raters used the rubric in evaluating 103 student coding assignments submitted for the Hangman lab. In order to maintain more natural results, raters were given limited instructions on how to grade. Rater 1 was a first year college student majoring in computer science and enrolled in an introduction to programming course. She took visual basic in high school, but only completed 3 months of her college programming course. For the purpose of this evaluation, the researchers considered Rater 1 a computer science novice that would be on par with the high school BJC teachers as most of the BJC teachers had limited experience in computer science training. Rater 2 obtained a Master's degree in computer science and taught a middle school computing club for over seven years. The research team considered Rater 2 a computer science expert during this evaluation process.

The 103 Hangman programs were collected from the 2014-2015 school year; 41 of them from a

**Table 4.1** Hangman Rubric developed from Learning Objectives and Essential Knowledge selected by Delphi panelists.

<b>Learning Outcomes</b>	<b>4</b>	<b>3</b>	<b>2</b>	<b>1</b>
LO 4.1.2 Express an <b>algorithm</b> in a language.LO 4.4.1 Develop an algorithm for implementation in a program.	Code cleanly organized and laid out to be read easily. Abstr. blocks are properly composed.	Much of the code is cleanly organized with certain parts disjointed/apart.	Half of the code is cleanly organized with the other half disjointed and separated.	Majority of code is broken apart and hard to follow by the reader.
EK 2.2.1B An <b>abstraction</b> extracts common features from specific examples in order to generalize concepts.EK 5.3.1A Procedures are reusable programming abstractions.	Code properly separated into deeper functions and abstractions. Different abstractions for each process: Word-List Display Word (etc.)	Majority of code is separated into different levels of abstraction with few functions not divided into new blocks for functionality.	Some of the code is abstracted properly, using different functions and code blocks to perform basic actions. Few functions are left non abstracted.	Little of the code is properly abstracted with the majority of functions left together.
EK 5.3.1F Parameters generalize a solution by allowing a procedure to be used instead of duplicated code.EK 5.3.1L Using <b>lists</b> and procedures as abstractions in programming can result in programs that are easier to develop and maintain.	Code blocks and abstract functions use list operators & parameters to increase program efficiency.	Majority of abstractions utilize parameters to be used for testing, some list operators are used for efficiency.	Some functions and abstractions utilize parameters and list operators are not used efficiently	Few abstractions utilize parameters.
LO 5.4.1 Evaluate the <b>correctness</b> of a program.EK 5.4.1C Meaningful names for variables and procedures help people better understand programs.	Naming of functions, including abstractions, accurately represents the problem being solved program is easy to read; program works.		Some functions named properly, others provide no meaning, program can be understood after a few read throughs, and mostly works	
EK 5.5.1D <b>Mathematical</b> expressions using arithmetic operators are part of most programming languages.	Code uses math and logic (loops, mod, etc.) to automatically monitor when the code begins, ends, and how it processes.	Majority of code uses mathematics to monitor and keep track of how it runs.	Some mathematics is used to automate the functions but there are uses of numerous manual function calls that could be replaced with logic to automate it.	Little or incorrect math is used to automate the program process and the program relies on manual function block calls.

high school CS Principles Honors class taught by a seasoned computer science teacher (15+ years teaching high school computer programming) and the remaining 62 from a college level introductory programming course. The college course was taught by a tenured computer science professor and labs were conducted by undergraduate teaching assistants trained by the professor (considered to have a similar subject level expertise as new high school BJC teachers). For the purposes of this study Honors CS Principles is categorized as the group 1 and the college course as group 2.

Furthermore to test whether or not the learning-based rubric is an improvement over the traditional performance-based rubric, the raters completed grading for both the learning-based rubric in Table 4.1, and a simpler performance-based rubric in Table 4.2. The performance-based rubric was created by a member of the research team following the same ITS schema as the Brick Wall performance-based rubric described in Chapter 3 [Cat16]. Each rater graded the full set of 103 Hangman programs using the performance-based rubric. Raters were instructed to leave comments on the grading sheet whenever a particular program raised questions as to what grade value should be given. An example comment might be "Student A completed the task incorrectly, but utilized all of the desired list functions. Student A was given partial credit." The evaluations of projects using these two rubrics were compared via range of end scores, inter-rater reliability for each rubric, and rater commentary and a debriefing interview for ease of rubric application. We were interested in seeing if individual student scores differed across rubrics. We hypothesized that the richer analysis provided by the learning-based rubric would lead to either lower or more varied scores compared to the performance-based rubric.

## **4.5 Results**

### **4.5.1 Performance-Based Rubric Results**

The raters graded programs separately, but were given similar instructions on how to use the rubric. Projects were graded on a 1-4 point scale, with 4 being the highest value. A composite score was given to each project by calculating the graded average for that particular project. When scored with a performance-based rubric, projects ranged from 1.87 to 4.0, all projects together averaged a score of 3.25. When broken down by course, projects submitted in group 2 averaged a composite score of  $M=2.99$   $SD=.99$ , and projects submitted in group 1 averaged  $M=3.55$   $SD=.93$ . These results can be seen in Figure 4.2.

Interestingly, when broken down by rubric category the two groups differed significantly in all but one category, readability. In all other categories, scores were significantly different, as seen in Table 4.3. The development of the learning-based rubric was motivated by the ability to further

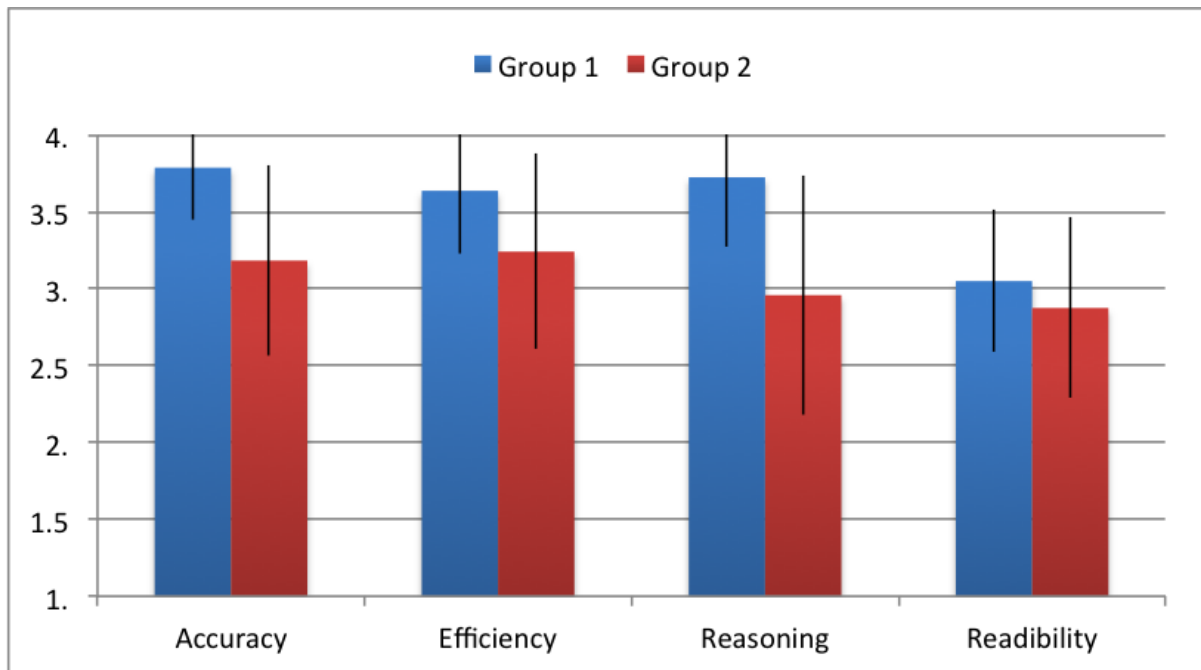
**Table 4.2** Hangman Rubric developed from performance-based rubric criteria in work by Cateté.

<b>Category</b>	<b>4</b>	<b>3</b>	<b>2</b>	<b>1</b>
<b>Accuracy</b>	The hangman game works properly (underscores where there are no letters, keeping track of the incorrect letters, generates a word every time, allows the right amount of turns, etc)	The hangman game mostly works and has all of the required components.	The hangman game does not work properly but has most of the required components.	The game does not work, and does not have a sufficient amount of the required components.
<b>Efficiency</b>	Uses lists and loops properly with very little repeated code.	Uses lists and loops, but still has some repeated code.	Uses lists and loops, but there is too much repeated code.	Does not use lists and loops.
<b>Rules and List Operators</b>	There is evidence of the use of logical concepts and appropriate use of abstractions and algorithms.	There is evidence of logical concepts or appropriate use of abstractions and algorithms.	there is inappropriate use of abstractions and algorithms.	Little or no evidence of the use of mathematical and logical concepts exists
<b>Readability</b>	The source code has been commented and is correct, logical, and easily readable.	Source code is correct, logical, and easily readable.	The source code is mostly correct, logical, and readable.	The source code is unclear, incorrect, messy, or incomplete.

expand the reasoning category.

#### 4.5.2 Learning-Based Rubric Results

After the use of the learning rubric to grade Hangman assignments, rater agreement was measured at 87.9%. When accounting for agreement by chance, inter-rater reliability measured as Cohen's kappa was satisfied at 74.8%. Similar to the performance-based analysis, a composite score was given to each project by calculating the graded average on the rubric categories for that particular project. When scored with a learning-based rubric, all projects together averaged a score of 3.57. When broken down by course, projects submitted in group 2 averaged a composite score of M=3.45 SD=0.82, and projects submitted in group 1 averaged M=3.69 SD=0.86. Data were also analyzed on an objective-by-objective basis for each class. Learning objectives related to the Hangman activity were categorized into five sections, four of which contained more than one knowledge description. For example, LO 4.4.1 *Develop an algorithm for implementation in a program* and LO 4.1.2 *Express an algorithm in a language* were combined into the algorithm assessment portion of the rubric.



**Figure 4.2** This figure compares projects evaluated by a performance-based rubric between group 1 and group 2.

Figure 4.3 shows the comparison of each learning objective and how well the students reportedly achieved it. In the algorithms category (Develop and express an algorithm) students scored 3.62/4. In the use of abstraction category (extracting common features to make reusable programming), students scored a mean of 3.65/4 informing us that students have a strong understanding of what abstraction is and how to implement it. The next category covers the use of parameters and list operators in the students' projects. As learning list operators and functions surround this lab activity, it is important that students can apply the knowledge they have been hopefully learning. Projects scored 3.37/4 on list operators. The naming and correctness category averaged 3.36/4 on student projects. The mathematical category is more broadly written to incorporate the different logical pathways created by students, as there are many ways to implement mathematical and logic expressions to solve a problem; projects averaged 3.61/4 on this.

We will discuss these results, and possible interpretations of them in Section 4.6.

**Table 4.3** Descriptive statistics for a comparison of means test on Group 1 vs. Group 2. (In each category Group 1 is the top line.)

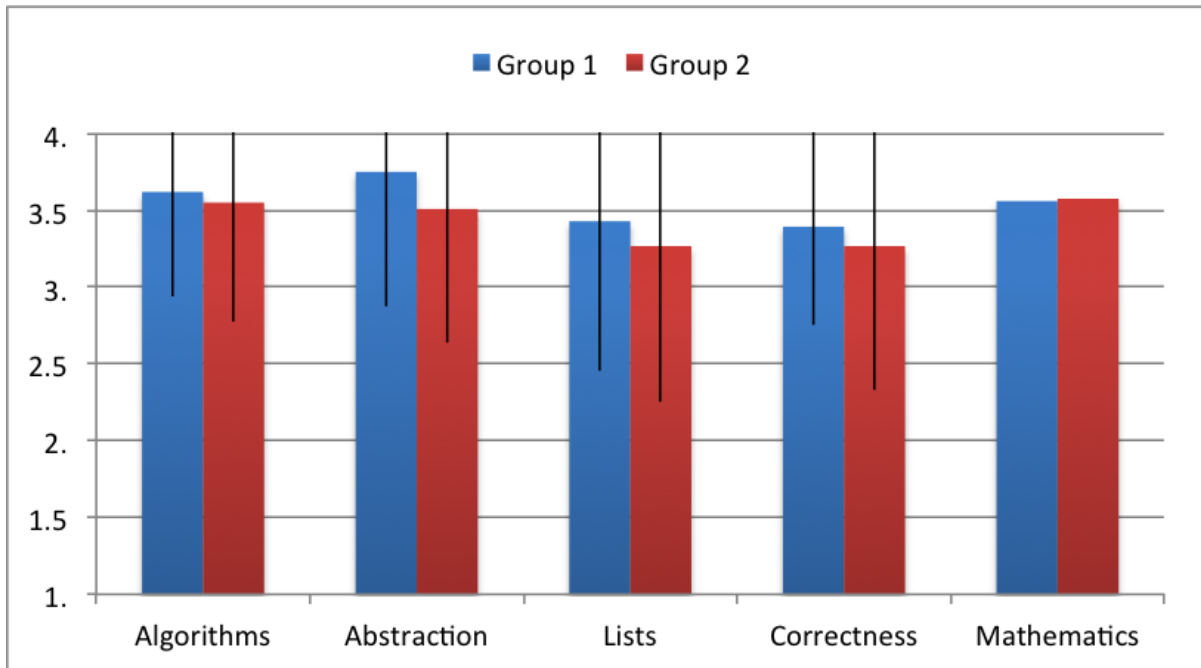
<i>Category</i>	<i>Sample size</i>	<i>Mean</i>	<i>Standard Deviation</i>	<i>Variance</i>	<i>p-level</i>
Accuracy	40	3.788	0.338	0.114	<.001
	60	3.183	0.617	0.381	
Efficiency	40	3.638	0.408	0.167	<.001
	60	3.242	0.634	0.402	
Reasoning	40	3.725	0.452	0.204	<.001
	60	2.958	0.777	0.604	
Readability	40	3.050	0.464	0.215	0.116
	60	2.875	0.587	0.344	

### 4.5.3 Between Rubrics

One aspect we wanted to look into was project evaluation between rubrics. We are interested in whether or not projects that score highly on the performance-based rubric also score high on the learning-based rubric. For this analysis we show the grade difference between average project scores on each rubric. The standard difference between grades is 0.6 points. When broken down into the separate categories on each rubric, with learning-based categories denoted as L and performance-based categories denoted as T, the categories with the best scores (on average) are Abstraction (L), Algorithms Composition (L), and Mathematics (Use of list/parameters) (L). The lowest scores occur (on average) in categories Readability (T), Reasoning (T), and Correctness/Readability (L). In Figure 4.4 we can see the correlation between grades of each student. The learning rubric based grades are on the X-axis, and performance-based on the Y-axis. It can be seen from Figure 4.4, that the student grades are not well correlated between rubrics,  $R^2 = 0.03$ .

## 4.6 Discussion

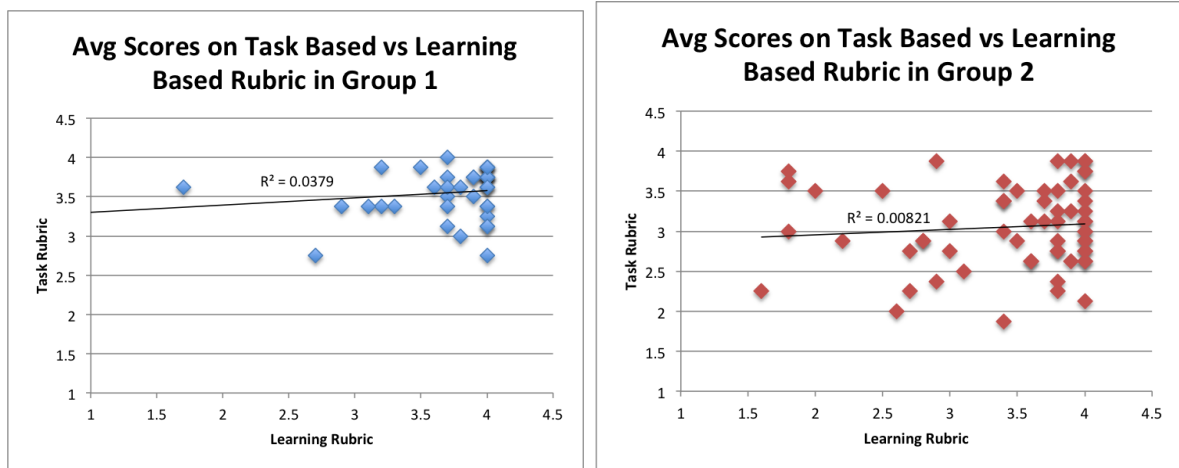
This study set out to evaluate the effectiveness of a new learning-based rubric developed by an expert Delphi panel. It was hypothesized that the new rubric would have strong inter-rater reliability and ease of use. We also surmised that the learning rubric would highlight aspects of the students learning that are either flawed or missing, resulting in lower scores on average. In this section, we will take a look at the results and discuss how they either support or contradict our hypotheses.



**Figure 4.3** This figure compares projects evaluated by a learning-based rubric between group 1 and group 2.

#### 4.6.1 Reliability

The first question in this study sought to determine if the learning-based rubric could be used reliably by both master and novice teachers. Given our two raters, the agreement level was a remarkable 88%. When accounting for the probability that our raters would agree by chance, the inter-rater reliability held strong, with Cohen's Kappa at 75%. What is even more surprising is that the raters did not display sufficient agreement on the performance-based rubric, scoring well below what is considered acceptable (Kappa <20%). A possible explanation for these results might be that the learning-based rubric had a written description or target goal in each category. When interviewed on her thought process while grading using the performance-based rubric, Rater 1 remarked, "It was kind of hard to tell whether a project was one grade or another, the descriptions seemed too close together in some categories." Rater 1 also stated, "I tried to give similar projects the same grades, but sometimes it was hard to remember what score I had given something because it was on the borderline." Without giving a clear standard, the rater had trouble knowing where the cutoff line was between performance-based rubric ratings. With the added goal descriptions in the learning-based



(a) Group 1 grade correlation

(b) Group 2 grade correlation

**Figure 4.4** Student score correlations across rubrics

rubric, we believe the raters were better able to gauge where the cutoffs were, leading to a higher level of consistency and inter-rater reliability.

#### 4.6.2 Ease of use

In this section, we discuss our findings on the rater experience while grading these programs. During Rater 1's interview, we inquired about her thought process while grading the performance-based rubric, and what made it so different from the learning-based rubric. When grading assignments on the performance-based rubric, first impressions went a long way for Rater 1. If the program area looked messy or it was not immediately clear where to start, Rater 1 automatically docked a point from the accuracy category. Additionally, if the program did not run correctly, meaning Accuracy did not score a 4, the rater was less inclined to give any other 4s. Rater 2 on the other-hand, would run the program first to see if it worked, then read through the whole program trying to figure out what was going on. Rater 2 also said that she gave the benefit of the doubt when grading future categories; one category was not dependent on another. Both raters agreed that programs that used the broadcast system instead of calling another function properly were given lower readability scores. Since we are developing rubrics for non-expert use, it is important that we discovered that a rubric that is typical for computer scientists to use leads to fuzzy categorizations on the part of the novice rater (Rater 1).

As explained earlier, the raters agreed much more frequently when using the learning-based

rubric. Rater 2 said that because the quality modifiers were consistent across categories, maintaining consistency in student grades was much easier. Rater 1 had a similar sentiment when she said that the learning categories had scores that were more definable. Additionally, as was briefly mentioned earlier, Rater 1 decoupled each category of the learning-based rubric. We believe this is due to the presence of separate learning goals. Rater 1 is more like a novice teacher, so higher reliability on the learning based rubric suggests that new CS teachers will be more likely to grade programs similarly to an expert in CS. For rater ease, raters found the learning-based rubric both easier to interpret and that it also allowed for more consistency between similar projects.

#### **4.6.3 Score distribution**

The third question in this research investigated how individual student grades compared across rubrics. We hypothesized that the more detailed focus on learning concepts would lead to highlighting lower student understanding of difficult concepts through lower or more varied grades. Contrary to our expectations, projects received lower grades more frequently on the performance-based rubric. Furthermore, grading with a performance-based rubric led to a normal distribution, whereas the learning-based rubric resulted in a left skewed curve. Despite the overall grade changes across rubrics, we also found that when separated into classes, there was no significant differences in performance when using the learning-based rubric. This is very interesting as the learning-based rubric is an expansion of the reasoning category on the performance-based category, which had most significant difference in student performance. This unexpected outcome could be due to the fact that the raters had low agreement on scores for the performance-based rubric. The disagreement could have offset the overall average for each category. Future analysis is needed to determine which aspect of the performance-based rubric caused the most discrepancies. Another explanation for this result is that the categories themselves focus more on what the students should have learned instead of the aesthetics of the programming artifacts. Whereas HS honors students might care more about the 'perfectionist' quality of their coding assignments, college students in an introductory non-majors computing class might put in only as much effort as needed. This would result in projects that look drastically different, but have the same functionality overall.

Note that we do not find the left-skewed distribution of scores on the learning-based rubric to be a disappointing outcome. On the contrary, the labs were designed to encourage complete novices to implement new computer science and computational thinking concepts without a strict focus on efficiency or typical measures of success in computing. We therefore find that this scoring may in fact be more encouraging for novice students, and enable them to see themselves as being capable of producing successful programs. This should in turn encourage students to further explore computer

science, rather than deterring them when their first attempts at code were not optimal from a computer science perspective. We will, however, continue to explore rubric usage and its impact on teacher understanding of learning objectives, and the impact of rubrics on student learning, particularly when the rubrics are shared with the students before the labs are due.

## 4.7 Conclusions

An initial objective of the project was to further explore the learning outcomes achieved by students when completing programming labs in the CS Principles/BJC course. In order to examine evidence of learning more closely, we selected a lab that was used frequently by BJC teachers, Hangman. We then created a learning-based rubric for the Hangman assignment that is closely aligned with the desired learning objectives and essential knowledge outlined in the CS Principles Course Framework. To create the rubric, we performed a Delphi study with a panel of BJC Master Teachers. The study resulted in a learning rubric with nine learning outcomes, as voted on by the expert panelists, organized into five different categories.

We then evaluated the learning-based rubric in comparison with a performance-based rubric created by the junior research team using ITS strategies outlined in Chapter 3. Both rubrics were applied in grading 103 student artifacts, 41 from a HS CS Principles Honors course, and 62 from a college level intro to computing course. In order to reflect the variance of typical BJC teacher background knowledge in computing, we selected both a novice and expert grader to do the assessments.

The results of this investigation show that using a performance-based vs learning-based rubric have varying effects on determining student project quality. Using the performance-based rubric led to not only inconsistent results between graders, but also a significant difference in the performance level of the student projects—one that might discourage novice learners. On average, the projects submitted by the group 1 performed significantly better in every category of the performance-based rubric except for readability, which showed no significant difference between groups. When using the learning-based rubric on the other-hand, graders had a very high level of agreement on student outcomes. Contrary to expectations given by the performance-based rubric, there was no significant difference in learning outcome grades between the two classes. This is surprising because the Reasoning category, which evaluates application of computational thinking on the performance-based rubric, has the largest gap between performance. The learning-based rubric is an expansion of and more refined implementation of the Reasoning category; we expected to see significant differences in at least one of the measured learning outcomes. We believe our expectations were colored by the fact that we are computer scientists, and more highly value code that is aesthetically more pleasing, and is computationally more efficient— and these values were strongly reflected in

the performance-based rubric. However, it may be unrealistic to expect students to achieve elegance when first learning computer science. A learning-objectives based rubric may more fairly reward novice students for learning of individual concepts.

Although at first glance, the difference in student performance scores between the two rubrics may call into question the validity of the learning-based rubric or the use of a novice grader - since the code artifacts were the same, but evaluated with different rubrics, isn't one correct and the other incorrect? We argue instead that the BJC labs are designed specifically to have a low threshold and a high ceiling - meaning that most students can do them, but that advanced students can still learn a lot from them. This means that by simply achieving the list of lab requirements, a student has demonstrated the intended level of competency for the AP CS Principles course. This explains the difference in rubrics. The performance-based rubric, developed to mirror a traditional grading scheme used in computer science, is biased towards giving more credit for more elegant, efficient solutions, which are not appropriate measures of success for novices. In AP CS Principles, it is more appropriate to award students full credit for meeting the course learning objectives, rather than penalizing students for not being more advanced to start with. We argue that measuring student programs based on whether students have met the intended learning objectives is a more objective goal that promotes equity in assessment.

Learning computer science is a complex domain where multiple learning objectives are often composed together, causing high cognitive load for novices. Removing bias from our rubrics and assessments might not just be needed for novice high school teachers, but for all of us, in order to make our courses more equitable and objective.

## CHAPTER

# 5

## STUDY 3: DELPHI METHODS IN CS PRINCIPLES RUBRIC CREATION

*(Best Paper Nominee) Cateté, V., & Barnes, T. (2017, June). Application of the Delphi Method in Computer Science Principles Rubric Creation. In Proceedings of the 2017 ACM Conference on Innovation and Technology in Computer Science Education (pp. 164-169). ACM.*

### **Abstract**

Growing public demand for computer science (CS) education in K-12 schools requires an increase in well-qualified and well-supported computing teachers. To alleviate the lack of K-12 computing teachers, CS education researchers have focused on hosting professional development workshops to prepare in-service teachers from other disciplines to teach introductory level computing courses. In addition to the curriculum knowledge and pedagogical content knowledge taught in the professional development workshops, these new teachers need support in computer science subject matter knowledge throughout the school year. In particular, these new teachers find it difficult to grade programs and labs. This research study uses two variations of the Delphi Method to create learning-oriented rubrics for Computer Science Principles teachers using the Beauty and Joy of Computing

curriculum. To perform this study we implemented (1) a heavy-weight, heterogeneous wide-net Delphi, and (2) a lower-weight, homogeneous Delphi composed of master teachers. These methods resulted in the creation of two systematically- and rigorously-created rubrics that produce consistent grading and very similar inter-rater reliabilities.

## 5.1 Introduction

This paper presents two variations of the Delphi Method used to develop well-defined, learning-oriented rubrics for programming labs taught by novice Computer Science Principles teachers using curriculum from *the Beauty and Joy of Computing* (BJC) [Uni15].

First, we give a brief overview of the current state of teacher preparation for Advanced Placement Computer Science Principles (AP CS Principles), followed by examples of rubric development for introductory computer science. We then describe the Delphi process or Delphi method, adapted from psychology, and its use in decision making. After a brief review of the Delphi process, we describe our application and evaluation of the Delphi in creating two separate well-defined rubrics. Finally, we compare the two Delphi process variations using inter-rater reliability, score distributions, and an overall cost-benefit analysis. We end our paper by discussing the merits of this methodology, and how to streamline the process for expanded use.

## 5.2 Background

As public demand for computer science in K-12 classrooms continues to grow, there is a large shortcoming in the number of well-qualified Computer Science Education teachers. In the United States, few colleges offer teacher education directly in Computer Science. Many education departments offer Technology Education, however, this track often pertains to the use of technology in the classroom and engineering practices, rather than computational thinking and computing.

### 5.2.1 U.S. K-12 Computing Teachers

With the lack of infrastructure to quickly prepare new teachers directly for K-12 computing, researchers and universities are holding professional development (PD) seminars and training to convert existing K-12 teachers into computing teachers [Bar16; Eri14; Goo14]. Attendees for this type of PD have diverse subject backgrounds ranging from Business to English and Language Arts [Eri14; Pri16]. In order to be successful in teaching a course, Shulman suggests that teachers should have subject matter knowledge, curricular knowledge, and pedagogical content knowledge (PCK)

[Shu86]. Teachers in the CS oriented PD sessions are being trained in curricular knowledge and pedagogical content strategies [Eri14; Pri16]. This support is a brief introduction and does not turn the novice computing teachers into content experts.

A 2016 study focused directly on measuring the computer science PCK of active K-12 computing teachers [Yad16]. The study showed that teachers felt confident in transmitting the associated computer science knowledge and understandings, however, teachers demonstrated difficulty in addressing student problems relating to programming errors. This difficulty understanding and debugging student code directly relates to the teachers' lack in expert content knowledge.

A lack of content knowledge leaves teachers unprepared to understand and identify computational thinking in code [Pri16], unprepared to assist students with programming errors [Yad16], and unprepared to create instructional rubrics for students to learn programming assignment requirements [Eri14].

### **5.2.2 Computational Thinking Rubrics**

A summative review by Stegeman highlights that while many introductory CS rubrics focus on similar aspects of code quality, they are very diverse in form as well as content [Ste16; Cat16]. Stegeman instead calls for a systematic approach to creating instructional rubrics that act as teaching tools to help students understand the requirements for an assignment to be successful. The instructional rubrics go beyond a summative scoring mechanism to also list verbal descriptions of the specific desired outcomes.

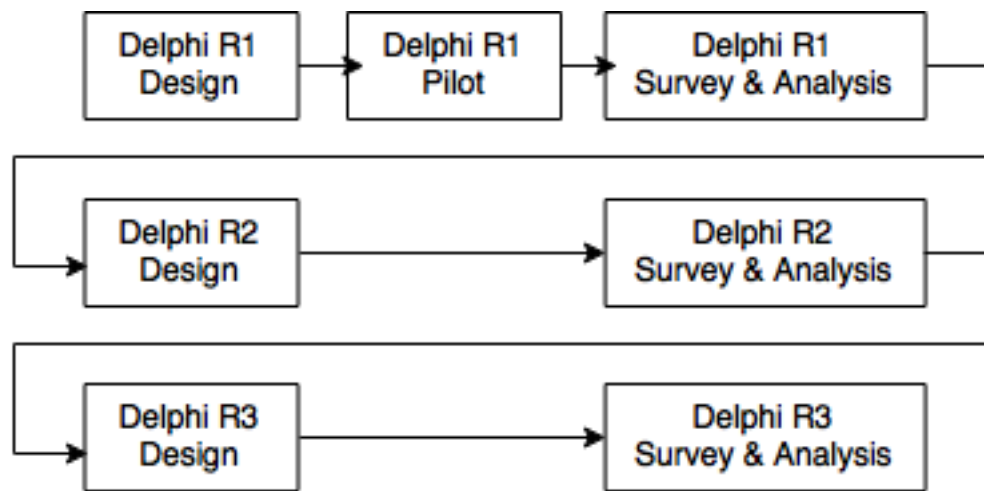
From the previously mentioned evidence, we believe that novice computing teachers need support in understanding and assessing student code through the use of learning-oriented rubrics. In order to systematically create these rubrics, we borrow the Delphi method from psychology and education.

### **5.2.3 Delphi Process**

The Delphi process is a technique originating in the 1950s to obtain the most reliable consensus of opinion from a group of experts using intensive questionnaires interspersed with controlled-opinion feedback [DH63]. As emphasized by Adler, expertise is tied to having knowledge and practical engagement with issues under investigation [AZ96]. As Delphi studies do not normally rely on random samples, these studies represent the best thinking and opinions of a group of people chosen for their special knowledge and experience [AZ96; Lec84].

The Classical Delphi is characterized by four pillars: anonymity of participants, iteration for participants to refine their views, controlled feedback informing participants of other perspectives,

and statistical aggregation of the group response to allow for quantitative analysis and interpretation of data [RW99]. As part of the Delphi Process, panelists complete a series of questionnaires which are analyzed and processed, with the results integrated into the next stage of survey rounds; Figure 5.1 outlines this process.



**Figure 5.1** A simplified version of the Delphi Process.

As outlined by Skulmoski, the typical Delphi method can vary greatly in number of participants, with the number of rounds usually three or four [Sku07]. As the number of rounds increases, so does participant drop off rate.

### 5.3 Methods

In order to move towards well-defined learning rubrics with a structured foundation, we consulted expert panelists through the implementation of a modified Delphi process. Throughout 2015, we carried out two separate Delphi studies; the first (StudyA) was a national poll with a heterogeneous sample of CS Principle stakeholders, the second (StudyB) was local to a professional development session with CS Principles Master Teachers (those who have taught 2+ years of CS Principles).

The subsections below describe how we systematically used the Delphi Process to solicit expert-chosen learning outcomes from the CS Principles Framework and applied them in the creation of learning-oriented rubrics for two popular BJC lab assignments. The final subsection 5.3.5, presents further details on how we evaluated the rubrics.

### 5.3.1 Lab Assignment Descriptions

Surveys show that most high school BJC teachers from 2012-2015 used the Brick Wall and Hangman Lab assignments in their classrooms, making these labs ideal candidates for our study.

BJC's Brick Wall lab was designed to demonstrate abstraction and the value of creating a function that can be called to perform the same task multiple times, and with different parameters. The main objective of the Brick Wall lab is to create a brick wall with an alternating pattern of bricks. In the assignment, two row types A and B are defined, where an A row is made up of whole bricks, and a B row starts and ends with half-bricks. Students are instructed to create one method that generates a brick wall, two separate methods that draw the two row types, and a method that draws an individual brick. These specific instructions are explained as levels of abstraction for solving the problem. The visual nature of the task and the clear and simple repetitive structure of a brick wall are affordances that should help make iteration and functions natural solutions to the problem.



Figure 5.2 Levels of abstraction for the Brick Wall assignment

The Hangman Lab is part of the Lists and Higher Order Functions units in the BJC curriculum [Uni14]. The lab was designed to combine tasks learned in exploring list operations such as 'map', 'keep', and 'combine.' This activity is more advanced than the Brick Wall lab and requires a better understanding of algorithms. The directions for completing the Hangman activity are below.

Imagine that you're writing a program to play Hangman. The program has thought of a secret word, and the user is trying to guess it. Write a display word block that takes two inputs, the secret word and a list of the letters guessed by the user so far. It should say the letters of the secret word, spaced out, with underscore characters replacing the letters not yet guessed. See Figure 5.3 below for an example.



Figure 5.3 Block displays a secret word 'Hangman' style.

### 5.3.2 Panelists

In order to develop a rubric with well-determined learning objectives in StudyA, we recruited a heterogeneous sample of 8 university Computer Science professors as well as 10 high school computing teachers through the SIGCSE mailing list. Participants were equally split 9 female and 9 male. Five participants indicated that they were involved in the development of either BJC or CS Principles and 9 participants had taught at least one year of CS Principles in their college or high school. These participants were considered expert panelists through their many years of teaching and researching computer science.

StudyB was comprised of 9 Master Teachers taking professional development to learn how to train new BJC teachers. Each participant was a current high school teacher with 2+ years of past experience teaching CS Principles; six female, three male. The primary courses taught by the panelists were: business management, mathematics, physical science, and computing. Two of the panelists have advanced degrees in teaching, two have Masters degrees in their subject field, and one of the teachers is National Board certified. Table 5.1 shows a summation of the Delphi metrics.

**Table 5.1** Summary of metrics used in Delphi Process implementation.

Delphi Process	# of Participants	Gender	Background	Study Rounds
StudyA: BrickWall	18 SIGCSE members	9 Female,	4 Tenured Professors	1 - Select and rank
		9 Male	2 Associate Professors 10 High School Teachers 2 Others	2 - Rate agreement 3 - Importance and code samples
StudyB: Hangman	9 BJC Master Teachers	6 Female, 3 Male	All 2+ years teaching BJC	1 - Select and rank 2 - Rate agreement 3 - Importance and code samples

### 5.3.3 Survey Rounds

StudyA and StudyB followed a similar implementation of the Delphi Process when it came to study rounds. We implemented a three round survey requesting expert panelists to indicate which learning objectives from the CS Principles Framework [Col14] best applied to the given lab.

In Round 1, panelists were given a questionnaire containing each of the learning objectives related to Creativity, Abstraction, Algorithms and Programming listed in the AP CS Principles Framework. Data and Information, the Internet, and Global Impact learning objectives were not included due to their indirect relation with the selected programming labs. Panelists were instructed to select as many objectives as they thought applied to the particular lab, and were additionally required to

indicate their top five choices for most relevant learning objectives. Once the surveys were completed, the research team compiled the lists keeping the top 80% of learning objectives indicated by frequency and any lesson objective that made a panelist's top five. The research team analyzed the survey results and proceeded to Round 2.

The Round 2 survey focused on rating the aforementioned highest ranked learning objectives. The survey was composed of Likert-based questions where each panelist had to rate how strongly they agreed that a particular learning objective pertained to the given lab assignment. The objectives were rated on a scale of 1 (Strongly Disagree) to 5 (Strongly Agree). Panelists then had the opportunity to indicate which Essential Knowledge components, discrete learning statements that make up each larger learning objective, were most relevant. Indicating essential knowledge components was important for broader learning objectives that had up to ten different sub-goals. The research team analyzed the results, carrying forward learning objectives and associated essential knowledge components rated 4.25 or above.

In the Round 3 survey, panelists were shown each objective rated 4.25 or above and given the option to contest any learning objectives they saw unfit. The uncontested learning objectives and associated essential knowledge components were thereby chosen by consensus from a panel of experts as being the most representative of the learning goals and objectives afforded by the given lab assignment.

The Delphi Process carried out by the research team differed from the Classical Delphi in that in the second half of Round 3, panelists were requested to submit their own demonstrations of the learning objectives in mock student coding samples. Panelists selected up to three learning objectives or related essential knowledge components and created a high, medium, and low level demonstration of that skill. Code samples could be submitted via Snap! source code [Uni17], program screenshot images, or text explanations with varying degrees of student understanding and misconceptions. Additionally, panelists were given sample student programs and asked to select the top three learning goals demonstrated by that sample and explain why the code demonstrates those goals in short response form. The second half of Round 3 served to bridge the gap between computational thinking learning objectives applied to writing vs. to code.

### **5.3.4 Delphi Application**

StudyA resulted in the selection of five learning objectives and eight essential knowledge components pertaining to the BJC Brick Wall lab assignment. StudyB resulted in the selection of five learning objectives and six essential knowledge components pertaining to the BJC Hangman lab. Each set of learning goals were then grouped by topic and split into five distinct rubric categories with one to

**Table 5.2** Sample Parameters category on learning-based rubric for Hangman lab.

EK 5.3.1F Parameters generalize a solution by allowing a procedure to be used instead of duplicate code. EK 5.3.1D Procedures have names and may have parameters and return values.				
4	3	2	1	0
Code blocks and abstract functions use parameters to increase usability and are easily changeable for testing.	Majority of abstractions utilize parameters to be used for testing.	Some functions and abstractions utilize parameters to allow greater functionality.	Few abstractions utilize parameters.	Abstractions do not utilize parameters.

two associated learning goals. Each category was given a simple name as well as the learning-based descriptions associated with them. See Table 5.2 for an example.

The coding samples generated by the Delphi panelists were also paired with the rigorously developed rubrics to show novice BJC teachers the relation of written learning objectives to the learning objectives as seen in code. A sample of this pairing is as follows: given Figure 5.4, a panelist has selected EK 5.5.1A *Numbers and Numerical concepts are fundamental to programming* as third in level of importance for relating to the code sample. Furthermore, the panelist explains, “The student has created algebraic expressions that generalize numerical concepts necessary to the problem’s solution. S/he has clearly demonstrated knowledge of the concept.”

```

+ Draw Brick Wall With rows # + rows with bricks # + bricks +
each all of length length # + and height height # + and mortar +
thickness thickness # +
script variables x y
set x to
length x bricks + bricks - 1 x thickness / 2
set y to
height x rows + rows - 1 x thickness / 2
go to x: x y: y
for i = 1 to rows
if i mod 2 = 1
Draw brick row A with bricks bricks of length length and height
height and mortar thickness thickness
else
Draw brick row B with bricks bricks of length length and height
height and mortar thickness thickness
change y by 0 - height + thickness
go to x: x y: y
  
```

**Figure 5.4** A portion of the code sample critiqued by Delphi Panelists in Round 3.

### **5.3.5 Rubric Evaluation**

We tested the newly created rubrics with three university computing majors and two BJC Master Teachers. Participants were given from 60 to 103 coding samples and asked to grade each one using the same criteria.

#### **5.3.5.1 Participants**

The university student graders were comprised of one graduate student, one sophomore, and one freshman all majoring in computer science. The graduate student was selected for their expert content knowledge in computing, representing the ideally experienced CS teacher. The sophomore was a less experienced computing major and demonstrated modest programming knowledge. The freshman had taken an introductory programming course in high school, but had not completed the first level computing class in university representing the novice CS Principles teacher.

The two BJC Master teachers were selected based on their past performance in professional development training as well as their willingness and eagerness to improve support materials for BJC. One teacher had successfully taught two sessions of BJC in their local high school, the other had taken the professional development course several summers in a row and now leads their own training sessions. The second master teacher, in addition to teaching at their local high school, was also enrolled in distance education computing courses to strengthen their content knowledge.

#### **5.3.5.2 Data Corpus**

We collected 103 Hangman programs and 94 Brick Wall programs from three BJC implementations in the 2013-2014 school year. Two of the implementations were completed as a high school CS Principles Honors class, the third as a university introduction to computing course for non-CS majors.

The two high school classes were taught by the same teacher, a seasoned computer science teacher with 15+ years of experience teaching computer programming and AP Computer Science A/AB. The teacher taught her CS Principles honors class using the BJC curriculum developed and shared by UC Berkeley. Students were given time in class to work on lab assignments collaboratively, any work not completed in the allotted time was assigned as homework. Thirty-nine Brick Wall assignments were collected between the two high school sections and forty-one Hangman assignments were collected. The difference in assignment counts is attributed to student absences.

In contrast to the two high school classes taught by an experienced AP Computer Science teacher, the college course had seven separate lab sections facilitated by five undergraduate teaching assistants (UTAs) under the tutelage of a tenured Computer Science professor leading the lectures.

The five UTA majors were: computer science, civil engineering, materials science, paper science, and textile technology. The UTAs had minimal prior teaching experience and no similar computing course. The course professor led the UTAs through the labs the week before students did them, in much the same way that high school BJC teachers keep one step ahead of their students when they teach a new class for the first time. Fifty-five usable Brick Wall assignments were collected between the seven lab sections and sixty-two Hangman assignments were collected. Two Brick Wall assignments were unusable due to improper submission format.

Researchers noted the difference in background and teaching experience of the many class instructors and felt that the UTAs portrayed an accurate representation of the range of active BJC teachers. The level of teaching experience and quality would be reflected in the submitted student assignments, therefore representing a wide variety of program implementations and demonstrated learning outcomes. A compilation of submitted assignments and associated graders is shown in Table 5.3.

**Table 5.3** A breakdown of rubric grading metrics.

	Brick Wall Lab	Hangman Lab
High School	39 projects	41 projects
College	55 projects	62 projects
Student Graders	Grad Student, Sophomore	Grad Student, Freshman
Master Teachers	N/A	A: 102 graded, B: 60 graded

Hangman assignments ranged from fully animated and functional games to haphazardly organized and occasionally working single method implementations. The Brick Wall projects ranged from completely accurate walls with numerous parameters, to one brick telling a joke to a circle.

### 5.3.5.3 Grading

In order to evaluate the effectiveness of the newly developed rubrics in supporting consistent and meaningful grading results, we had pairs of raters grade each assignment. A graduate-undergraduate pair graded both assignments and the master teacher pair graded the Hangman lab. Teacher B was only able to commit to grading half of the assignments due to heavy course load and ended up grading 60/103 of the Hangman projects.

In order to maintain more natural results, graders were given limited instructions on how to grade. They were instructed, however, to leave comments on the grading sheet whenever a particular

program raised questions as to what grade value should be given. An example comment might be “Student A completed the task incorrectly, but utilized all of the desired list functions. Student A was given partial credit.”

## 5.4 Results

The aforementioned methods and research culminated in the creation of two rigorously developed learning-based rubrics hand-tested on a data set of 100 samples each by paired graders. When tested for inter-rater reliability the student-graders achieved a satisfactory .83 agreement on the Brick Wall lab assignment and .78 on the Hangman lab assignment. In the student-graded assignments, the grader acting as a computing expert remained consistent, the changing factor was the level of computing content known by the novice undergraduate student. When tested for inter-rater reliability the master teachers achieved a satisfactory .79 agreement on the Hangman lab assignment.

When broken down by course, Brick Wall projects submitted in the high school class averaged  $M=3.63$   $SD=0.78$  and projects submitted in the college class averaged a composite score of  $M=3.06$   $SD=0.81$ . Table 5.4 shows average score by rubric category.

**Table 5.4** Avg. Brick Wall project scores [0-4] using a learning-based rubric.

	Algorithms	Abstraction	Parameters	Correctness	Mathematics
High School	3.53	3.80	3.60	3.75	3.50
College Class	3.10	3.25	3.45	3.00	2.50

When broken down by course, Hangman projects submitted in the high school class averaged  $M=3.71$   $SD=0.86$  and projects submitted in the college class averaged a composite score of  $M=3.45$   $SD=0.82$ . Table 5.5 shows a breakdown of average score by rubric category.

**Table 5.5** Avg. Hangman project scores [0-4] using a learning-based rubric.

	Algorithms	Abstraction	Lists	Correctness	Mathematics
High School	3.80	3.92	3.63	3.51	3.70
College Class	3.56	3.52	3.26	3.34	3.60

## **5.5 Discussion**

In this discussion we examine the uses of the Delphi process, the effectiveness of the resulting rubrics, and strategies for streamlining the process for quicker and larger scale rubric creation.

### **5.5.1 Delphi Process**

When reflecting on the dual implementations of the Delphi method, we find that the overall outcomes are the same. The main differences between the national poll and the PD group are the time needed to complete the study and the amount of participant attrition.

The national poll took 11 weeks to complete from the initial release of the first survey to the close of the final-round survey. Participants were given three-week intervals to complete each survey round interspersed with one week for the research team to analyze results and create the next survey. As the time and participation requirements increased, so did the attrition rates. Forty-nine participants started the first-round survey, but only fifteen completed the final-round survey. Since much of the SIGCSE mailing-list contains active university and high school teachers, we believe that time available to participate fluctuated with holidays and teaching schedules.

The second Delphi study took one week to complete from the initial release of the first survey to the close of the final round survey. As panelists in this group were selected from a professional development summer training session, they were given designated time throughout the week to complete each round of the survey. Additionally, none of the teachers were actively teaching summer classes, so the panel was less distracted. The homogeneous group of participants all had equal stakes in the outcomes this research, as these newly created rubrics would be added as support material to the BJC curriculum. Additionally, participants were committed to attending the PD so access and attrition were not an issue.

The attrition and commitment levels demonstrated by both panels is consistent with findings presented in Skulmoski [Sku07].

### **5.5.2 Effectiveness of Rubrics**

Although developed by separate groups, the reliability of the rubrics turned out to be similar and satisfactory. Each of the three paired graders were able to achieve a high level of inter-rater reliability ( $> .70$ ). Additionally, the relationship in overall measured performance of the two data sets is as expected. The Honors CS Principles high school students, when measured by the rigorously developed rubric system, outperformed the sophomore and junior-based students taking an Introduction to Computers course for non-CS majors. As indicated previously, the Honors students were being

taught by a highly-qualified teacher with years of experience in teaching computer science courses, where as the college students completed most of their assignments in lab sessions facilitated by novice computing undergraduate TAs.

### **5.5.3 Cost-Benefit Analysis of Methods**

When looking through the lens of practicality, we are able to see a clear winner in regards to implementation of the Delphi. Polling the smaller more concentrated group of Master Teachers was much faster and just as effective as polling the larger heterogeneous SIGCSE-list. Less time was required on administrative part, as fewer reminders had to be mailed out. Panelists also had designated time set aside to focus on the study.

Furthermore, when selecting graders to evaluate the rubrics, we found minimal difference in using early year computer science majors and BJC Master Teachers. Both groups measured similar results while using the rubrics. When looking at availability, we find that the student graders were more reliable and able to take on extra work.

## **5.6 Conclusions and Future Work**

The first goal of this study was to explore the use of the Delphi Method in creating a rigorously-vetted and systematic basis for learning-oriented rubrics. We focused our development on rubrics for two of the most popular BJC labs taught by first and second-year high school BJC teachers. Although these teachers may have many years of teaching experience in other subjects, their subject matter content knowledge in programming and ability to recognize computational thinking in code needs support.

As part of the Delphi process we surveyed two separate groups of expert panelists: members of the SIGCSE mailing list (both professors and K-12 teachers), and BJC Master Teachers. Both groups completed 3 rounds of consensus-building surveys and rating systems to determine the most-important and most-relevant learning goals associated with the respective lab assignments. Using these learning goals, we created two unique rubrics with five categories of learning goals comprised of Learning Objectives and Essential Knowledge from the AP CS Principles Framework [Col14].

The second goal of this study was to evaluate the use of the new learning-oriented rubrics by sample graders. There are two limitations to this study design. The first one is that the Delphi method uses experts and if different experts were surveyed we may have different outcomes. The second limitation is similar in that it depends on just a few raters and different raters may have different results. In this case, we found that each pair of graders was able to establish inter-rater reliability and

the overall measurements in student performance reflected the differences in student backgrounds and teacher proficiency.

When reflecting on this study, we find that a smaller Delphi panel with active BJC teachers shows minimal drawbacks when compared to a larger heterogeneous panel. Additionally, we find that when using the rubrics, lower classification CS majors perform comparably to Master BJC Teachers. Practically, there are still major drawbacks in holding a full Delphi Panel, such as locating proper stakeholders with time-availability, knowledge on the subject, and willingness to participate.

In future studies, we will replace the master teacher panel with upper classification computer science majors, in order to create comparably rigorous rubrics at a faster pace and at scale. It is our hypothesis that a small team of subject-literate computer science majors (comparable to BJC Master Teachers), when given the BJC curriculum and CS Principles Framework, will be able to draw comparable connections in learning goals for lab assignments. We also believe that using an expedited form of the Delphi Process as we did in study 2 ended up being very similar to the nominal group technique (NGT) [AZ96]. As such, we will use NGT to continue to ensure an objective consensus by team members in future implementations.

## CHAPTER

# 6

## STUDY 4: A STREAMLINED APPROACH TO THE SYSTEMATIC CREATION OF RUBRICS FOR CS PRINCIPLES

### 6.1 Introduction

Recent developments in the spread of K-12 Computer Science education have heightened the need for professional development and support materials for high school teachers new to the computing classroom. Often, these novice teachers are given a pre-built curricula from a third party entity such as Code.org [Cod; Col17] or an academic institution [Uni15; Ute]. One of the AP Computer Science Principles curricula has gone through numerous iterations of redesign while simultaneously being available to high school teachers for active use in the classroom. Throughout the redesigns, many of the student lab assignments were distributed to teachers without rubric support. Although the teachers using the curriculum take part in a multi-week professional development program, there is still room for improvement in their understanding of computational concepts as displayed in student programs. Without proper rubrics, the teachers were unsure of the most important goals of each lab, and how to know if students met those particular goals in their programs.

In previous research [Ste16; Cat16; CB17], several systematic approaches to creating rubrics have been presented, however, they are not cost-effective for the frequent restructuring of CS activities. In this research, we present a streamlined approach to creating rubrics that stems from the Nominal Group Technique established in educational psychology. We hypothesize that this alternative method will lead to improved practical use in designing rubrics, than of previous researchers.

The next section of this paper will examine the different techniques used for developing rubrics in beginner computer science courses. It will then go into a brief introduction of techniques for improved methods in group decision making. The remaining part of this paper explains our new approach and the resulting outcomes.

## **6.2 Background**

Research on systematic rubric development has increased in the past several years reflecting the need for increased instructional support for inexperienced graders [Ste14; BD16; Yua16; CB17]. This rise in inexperienced graders is present in both university and secondary level schools. An increased number of teaching assistants are needed for growing CS1 class sizes and more K-12 teachers are transitioning into computing from other disciplines.

The most common methods for systematically generating these rubrics has been a mix of surveys of the field [Ste14; Cat16] and various group decision making techniques [Ste16; Yua16; CB17] which require a large amount of effort and administration. The next two subsections will examine the current methods for creating rubrics and then introduce an alternative methodology for timelier group decision making.

### **6.2.1 Rubrics in CS**

Both Stegeman and Cateté's early research attempts at systematically creating rubrics focused on in-depth literature reviews of the field [Ste14; Cat16]. Stegeman's effort examined and cross-referenced code quality handbooks for software engineering, whereas Cateté investigated literature on auto-graders such as ASSYST. Stegeman et al. paired down their generated list of over 400 quality statements down to just 9 criteria using instructor interviews and feedback. Cateté's list was filtered down to 5 criteria that were general enough to apply towards a blocks based programming language. Cateté then converted these criteria into a 4 point rubric which was iteratively revised through student and instructor testing until sufficient inter-rater reliability was achieved. Cateté found the the rubric generated auto-grading criteria focused primarily on task-completion and did not highlight learning goals which would lead to better teacher support and expected student outcomes[Cat16].

The later attempts at creating rubrics relied on expert generated rubrics [Yua16], some using group decision making techniques [Ste16; CB17]. Yuan et al. had an experienced professor create an 'expert' rubric that would be used by novice graders participating in a crowd-sourcing program. 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. These results only measured perceived helpfulness by students and do not focus on validation or reliability of the rubric which is a focus of Stegeman [Ste16] and Cateté's [CB17] work.

Stegeman et al. revised their initial rubric scheme (2014) through a 3-part 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 such that a student could understand "what good performance on a task is; how their own performance relates to good performance; and what to do to close the gap" to improve their score [Ste16]. 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.

Cateté's revisit to rubric creation pivoted from the task-based rubric developed using auto-grader literature to learning-based rubrics generated through Delphi surveys [CB17]. Cateté performed a national and local modified Delphi survey to generate two separate rubrics. During this process expert computer science 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. Only criteria with the highest level of consensus were included in the final rubrics. Consensus was achieved through 3 rounds of iterative surveys spanning up to 11 weeks. The generated rubrics were then tested for inter-rater reliability between novice and expert graders, where moderately high levels ( $k > 0.7$ ) of reliability were found.

In each of the presented rubric development methods, a large amount of time is needed to generate each rubric. Furthermore, many of these rubrics rely on the presence and assistance of one or more experts in the field. These methods limit the scalability of rubric creation, which is critical during this time of increased infusion of computing courses into K-12 schools. We believe we can improve on these methods while maintaining systematic rigor by streamlining the Delphi method used in Cateté 2017 and instead utilizing the similar, but more time efficient, Nominal Group Technique.

## 6.2.2 NGT vs. Delphi

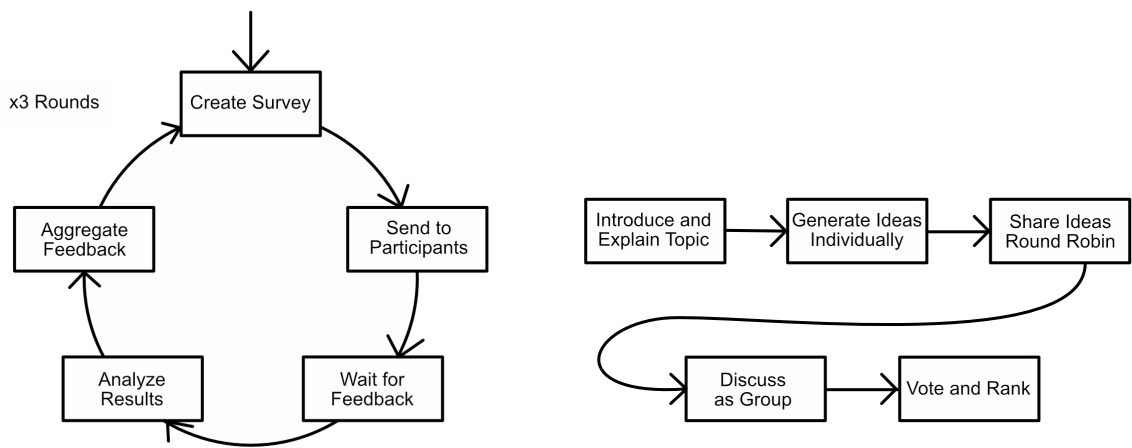
In Cateté's 2017 report, the Delphi method was shown to be a rigorous method for creating well-defined learning-based rubrics, however, the report also listed such drawbacks as time commitment, administrative involvement, and resource acquisition [CB17]. Research has proven the Delphi method to be useful for obtaining consensus from a group of expert opinions by relying on four pillars: anonymity, iteration, feedback, and aggregation [Sku07]. These pillars allow for asynchronous group decision making by geographically dispersed participants, but also require a large amount of deliberation between idea solicitation and aggregation. Another technique in Educational Psychology that seems better fit to the situation of rubric creation and learning objective generation is the Nominal Group Technique (NGT).

NGT and Delphi surveys are both methods for group brainstorming by a structured 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 [Gus73]. Similar to the Delphi method, NGT is a product of 1960s social psychology and is used as a bridge between researchers and practitioners [Del86]. Additionally, both methods rely on the belief that consensus by experts can generate a strong, effective, empirical generalization [Pow03]. The primary difference between these two techniques is that NGT forgoes the Delphi pillar of anonymity and instead chooses to benefit from the time saving effectiveness of co-located participants. Figure 6.1 illustrates the two processes.

While participants in the Delphi method are geographically dispersed, participants in NGT are co-located and talk directly with the facilitator and one another. The in-person interactions are advantageous in that participants can build rapport with the facilitator and feel a sense of accomplishment as results are seen immediately after the session [HH12]. A caveat to this in-person interaction however, is that the facilitator must refrain from overriding a diversity of opinion to create an artificial consensus. As doing so calls into question the validity of the final results [Tei06].

The modern NGT protocol has been framed by researchers into the following five steps [Pot04; McM16; HH12]:

1. *Introduction and explanation:* The facilitator welcomes the participants and explains to them the purpose and procedure of the meeting.
2. *Silent generation of ideas:* The facilitator provides each participant with a sheet of paper with the question to be addressed and asks them to write down all ideas that come to mind when considering the question.
3. *Sharing ideas:* The facilitator invites participants to share the ideas they have generated. The



(a)

(b)

**Figure 6.1** (a) The Delphi Process hinges on iterative cycles of survey generation and response aggregation. (b) The Nominal Group Technique encourages participants to write ideas down individually prior to sharing with the group.

ideas are usually written down on a board for all other members to reference. The round robin process continues until all ideas have been presented. No debates should happen during this time.

4. *Group discussion*: Participants are invited to seek verbal explanation or further details about any of the ideas that colleagues have produced that may not be clear to them. The facilitator's task is to ensure that each person is allowed to contribute and that each idea is discussed.
5. *Voting and ranking*: This involves prioritizing the recorded ideas in relation to the original question. Following the voting and ranking process, immediate results in response to the question are available to participants so the meeting concludes having reached a specific outcome.

In the next section, we present our methods for building on the Nominal Group Technique to streamline the rubric-making process to create a large suite of rubrics.

## 6.3 Methods

A number of techniques have been used to create well-defined and meaningful rubrics for introductory computing courses. While these rubrics were built with advanced theoretical bases in mind, they are not time-efficient for the quickly evolving CSP coursework. In order to maintain the rigor of the learning-based rubric while improving upon the implementation costs of the Delphi method, we implemented a modified NGT process. One advantage pointed out by Cateté [CB17] is that the localized version of the Delphi worked much smoother with all participants meeting in person for the week. Unfortunately, gathering a group of K-12 computing teachers during the school year is not easy to do, particularly since there are never more than a few computing teachers in a single district. In this section, we describe how we modified the variables listed in Cateté 2017 and refactored the process to closer align to NGT which we believe fits a more practical development process.

### 6.3.1 Training new 'Masters'

Although Master teachers are ideal candidates for creating rubrics, it is not feasible to get them together for a long enough period to create all the rubrics. In Cateté 2017, it was suggested that a pair of Computer Science undergraduates were able to grade rubrics as well as the pair of Master teachers. Therefore, we chose to form a group of Computer Science undergraduates to serve as substitutes for Master teachers in our process. In this experiment, we used a team of four advanced Computer Science undergraduates in an independent research course in Computer Science. All of the undergraduates were in their last two years of the CS major.

To prepare the team, we introduced CS Principles and rubric development during two 90 minute training sessions. First, we introduced the new team to the course and the AP CS Principles Framework. We walked the team through the seven big ideas of AP CS Principles (Algorithms, Abstraction, Creativity, Big Data, the Internet, Programming, and Collaboration). The AP CS Principles curriculum framework includes learning objectives and essential knowledge components for each big idea; these form the basis of our learning-based rubrics. After this introduction, we taught the team about the differences in analytical and holistic rubrics as well as the difference between task-based and learning-based rubrics as identified by Cateté et al.

Once the junior research team demonstrated basic understanding of the computing concepts and rubric techniques through verbal questioning and examples, we instructed them in their task of creating new CSP rubrics for the active lab assignments in the curriculum. The team was instructed to make both task and learning-based rubrics, however, this paper only focuses on the latter.

### **6.3.2 Streamlining the Process**

In order to systematically create the new set of learning-based rubrics, we implemented a modification of the 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 the undergraduates into teams of two, we distributed the course units and major lab assignments. We defined a “major” lab assignment as one 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 essential knowledge components they felt best matched the assignment. They were instructed to focus particularly on objectives dealing with programming, algorithms, and abstraction since they were the most relevant big ideas for labs.

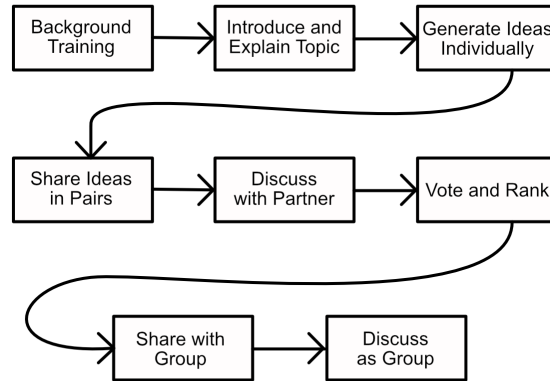
Once each person had a list of learning objectives, they shared and discussed it with their partner. Any objectives that overlapped were automatically included, and those that did not overlap were discussed until the pair could come to an agreement. After each pair of researchers compiled a list of learning 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 corresponding learning objectives, the team was instructed in how to cluster related objectives and set anchor points. For each category of the rubric, the team was instructed to come up with a simple description of one or two words, and then a set of 2-3 objectives. The next step in developing the rubrics was for the team to create anchor points, or benchmarks.

These markers would indicate a completed version of the assignment followed by textual descriptions for the rubric levels of above expectations, below expectations, and needs improvement. A score of zero was reserved for no attempt made. Each pair created corresponding rubrics for the lab assignments they investigated earlier. Our process is illustrated in Figure 6.2 and can be compared to the original NGT methodology in Figure 6.1b.

### **6.3.3 Quality Assurance**

Once the team had completed their rubrics for the labs, we had an external fifth undergraduate researcher go through and unify the rubrics. As each set of rubrics was created by a separate pair, there were slight differences in their wording or phrases. We used a fifth researcher, trained in a similar fashion, to read through all of the rubrics and align the goals and anchors where needed. For example, if pair 1 labeled a category clean-code and pair 2 labeled a similar category cleanliness this fifth person would go through and relabel categories to match. This was also true for aligning anchor values such as ‘some abstract functions’ versus ‘a few abstract functions’.



**Figure 6.2** Our Modified Nominal Group Technique pairs participants prior to group discussion.

In order to create rubrics more quickly while still maintaining a systematized procedure influenced by theory, we augmented the Nominal Group Technique as described by Delbecq and McMillan [Del86; McM16] to accommodate a more efficient work flow given the number of rubrics needed for a full curriculum.

## 6.4 Results and Discussion

After applying our modified NGT process a total of 32 learning-based rubrics and 34 task-based rubrics were created that corresponded to each of the lab assignments in the seven CS Principles curriculum units. All 66 of these rubrics were created over a span of ten sessions each lasting between 60-90 minutes.

### 6.4.1 A Modified NGT

This study aimed to test whether or not we could improve upon methods for developing introductory computing rubrics. In order to accomplish this, we trained a 4 person team in the CS Principles curriculum complementing their pre-existing computer science skills. We further enhanced the team by giving them supplemental training in rubric development. By creating a localized team of ‘almost experts’ [Yua16], we were able to adapt the NGT process to be used for rapid rubric creation.

NGT relies on qualified participants to make expert judgments and opinions on a given topic.

**Table 6.1** Common learning objectives between between assignments with similar mechanics

Pascal's Triangle - Algorithms	Making a Forest - Algorithms
EK 5.5.1A Numbers and numerical concepts are fundamental to programming.	EK 5.5.1A Numbers and numerical concepts are fundamental to programming.
EK 4.1.1F Using existing correct algorithms as building blocks for constructing a new algorithm helps ensure the new algorithm is correct.	EK 4.1.1F Using existing correct algorithms as building blocks for constructing a new algorithm helps ensure the new algorithm is correct.
EK 4.1.1D Iteration is the repetition of part of an algorithm until a condition is met or for a specified number of times.	EK 4.1.1E Algorithms can be combined to make new algorithms.

Unfortunately, gathering multiple CS Principles teachers together for a prolonged period of time is not always feasible. Furthermore, not all CS Principles teachers have a deep background in programming or computing. Since the CSP course was just created with its first national exam in the United States in 2017, high school teachers have little experience using the CSP framework and little to no experience grading labs for this course. Most high school teachers who have participated in CSP professional development took one to five computing courses in their undergraduate degrees. We argue that 3rd and 4th year computer science undergraduate students have sufficient computer science expertise to understand novice CSP student programs, and that a focused workshop that introduces them to the CSP framework is sufficient preparation for rubric development.

Other modifications that we made to the NGT process includes inserting a shorter refinement step between a pair of participants before introducing ideas to the full group. These steps are similar to a think-pair-share [Kot13] methodology. First, we had pairs work together to create a list of learning objectives, grouped by category (e.g. algorithms, programming, abstraction) representative of the final rubric. By submitting full drafts of rubrics rather than submitting 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.

Another benefit to submitting whole rubrics was that the discussion time spent on an individual rubric was more focused, streamlining the total time spent on developing an individual rubric. The team would take about 20 minutes to create a learning based rubric, which required matching learning objectives from the CS Principles Framework to lab descriptions. This time varied depending on similarities between lab assignments, as some projects built on earlier ones.

### **6.4.2 Rubric Quality and Distribution**

After the initial research team created all of the rubrics, a fifth trained person combed through each of the rubrics for alignment of language, grading scales, and category names. Although much of the work dealt with formatting and minor text corrections, we did notice a trend in the rubrics. A few of the category titles were popular across multiple rubrics. For instance, each rubric had a category with a simple name, such as ‘algorithm’ which would include the overarching learning objective plus additional essential knowledge components tailored to the specific assignment. Table 6.1 shows two separate algorithm categories with similar learning objectives. This is interesting because it demonstrates a consistency in our team to recognize algorithmic thinking in the labs, but also because we can see that for similar programming mechanics across assignments, our team chose to evaluate them using similar measures. Although this is not a true measure of validity, it indicates consistency in the content of these rubrics.

After unifying each of the rubrics into a similar format and language, the rubrics were released to the active CSP teachers for immediate use through both an email list as well as through an on-line CSP teacher discussion forum. We asked one of the CSP professional development leaders to post the rubrics as a new resource, since the teachers were more familiar with, and thus responded more frequently to, this leader. The rubrics were made available through a bulk archive file or individually through a folder hosted online. After the rubrics were made available to the CSP teachers, there was a significant decline in new requests for rubrics. This led us to believe the teachers were satisfied with the rubrics created.

### **6.4.3 Limitations**

There are several limitations to this study. The primary limitation is that we cannot yet provide measures of validity or robustness for the rubrics. We plan to investigate these features of the rubrics in future work. Although they may not yet be validated, providing these rubrics quickly to new teachers helps them feel more confident in teaching this new CSP course, which is important in getting the class taught.

Another limitation to this experiment is that we were not yet able to collect formal feedback from teachers who have actually used the rubrics in their classroom. Given the distribution platforms used, it was not possible to collect data on number of downloads or how the rubrics were used. The CSP trainers collected some feedback during bi-weekly conference calls with the new CSP teachers. CSP trainers reported that new teachers just learning to teach the course “feel more comfortable about determining whether or not their students have learned the necessary skills.”

Similar to the previous limitation, we are unsure how many teachers used the rubrics for reference

versus those who used them in their classroom or distributed them to students prior to assigning lab work. As educators, we consider all of these to be positive (if unmeasured) outcomes. Teachers who use the rubrics as a reference can gain insight into how to convert CSP learning objectives into written descriptions of code. CSP teachers who use the rubrics for grading now have an objective rubric that should help them grade more fairly and consistently [AK09]. Students with access to the assignment specific rubric before grading occurs can plan their work with better knowledge of how it will be evaluated [Eug16].

Although we find limitations in the current presentation of our study, overall, these results indicate a positive direction for systematic rubric creation that is quicker and more feasible to implement than prior methods. During the modified rubric creation process, two separate CSP units in the curriculum were taken down and reorganized. Our research team was able to adjust around this and update the new rubrics within a week. While the reliability and validity of the rubrics is uncertain, having them available this quickly helps new teachers handle newly changed curricula.

## **6.5 Conclusions and Future Work**

This research suggests that a co-located team of trained advanced undergraduate computing majors can create rubrics for use by new computing teachers to assess learning objectives for an introductory Computer Science Principles course. Creating expert rubrics have taken researchers a significant amount of time through both survey of the fields and group decision techniques [Ste14; CB17]. Using the traditional Delphi method identified in Cateté 2017, it took the researchers 11 weeks to get the final version of one rubric. Using the modified in-person Delphi took much less time, but still utilized a group of expert teachers.

Partnering with a group of teachers, in-person, during the active school year, is much more difficult than during a summer PD session. Unfortunately, CS Principles and other K-12 courses are being updated on a rolling basis throughout the year. To address these challenges, we have trained a team of experienced computer science students on a CS Principles course in order to become ‘almost experts.’ We then used this team to create rubrics for a full CS Principles course using a modified NGT method which further streamlined the rubric making process. Our method decreased the time to make a rubric to a single session. In just 10 sessions, the team systematically created 66 rubrics (learning and tasked based), which have subsequently been released to active CS Principles teachers.

Further research will be conducted to evaluate the reliability and validity of these rubrics. Preliminary investigations into validity metrics has led us to believe that the validation criterion set out by Messick [Mes96] is more suited for high-stakes testing, and that Baartman’s Wheel of Competency

[Baa06] is better suited for evaluating low-stakes project based rubrics. We plan to apply Baartman's methods, and expect that the resulting reliability of the new rubrics will show minimal difference between experienced and novice graders as they were developed in a systematic way using clear and consistent language.

## CHAPTER

# 7

# STUDY 5: AN EVALUATION OF BJC RUBRICS WITH ACTIVE CS PRINCIPLES TEACHERS

## 7.1 Introduction

In this chapter, we extend our evaluations of CS Principles rubrics we developed in Chapter 6, which were created using a modified version of the Nominal Group Technique (NGT). We recruited active CS Principles teachers and current STEM Education university students to use our rubrics while evaluating student projects. In this study, we wanted to compare participants' ability to identify relevant learning objectives directly in student's code as well as evaluate how educators were able to use our rubrics consistently.<sup>1</sup>

---

<sup>1</sup>See Chapter 2 and Chapter 6 for more background information relevant to this study.

## 7.2 Methods

The priority for this study was to recruit newly active high school computing teachers and soon to be CS Principles teachers. We limited participation in the study to those had taught less than 2 years of AP CS Principles and who have not taught other computer programming courses including AP Computer Science A. We recruited from the CSTA mailing-list, and allowed cs teachers to share this research opportunity with their STEM educator peers. We also invited student participants from NC State's College of Education via emails sent out through Science, Technology and Math Education administrators. 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.

### 7.2.1 Participant Procedure

For the study, participants completed a pre-post survey with 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 sample coding but no rubrics.

Once the pre-survey was complete, participants were directed to part two where they would reflect on their evaluation strategies for the previous samples. Sample reflection questions include, "What metrics did you use to grade the assignments? Did you refer to the learning objectives for the course? If you did not take a look at the learning objectives, what metric did you use to gauge student learning?" After reflecting on their evaluation strategies, participants were provided with the list of Essential Knowledge items relevant to the prior labs. Participants were given the descriptions for three lab programming assignments and were to pair the appropriate Essential Knowledge (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!. The responses to the reflective questions and EK matching were not stored as these were practice activities.

After participants gained experience in grading Snap! student projects, they completed a final post-survey to conclude the study. In the post-survey, participants graded three samples for each of three different programming labs, highlighting 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 CT concepts.

### 7.2.2 Lab Assignment Descriptions

- **Beyond Binary** This lab extends the original decimal to binary lab assignment. In this version, students are to generalize the pattern for conversion from base 10 to base 2 with a 'base' block that takes the base as a second input. Support for bases above 10 are optional. The figure below demonstrates the code for converting decimal to binary. To convert this to another base, a student need only add another input parameter and replace all the 2's in this code with the new base.

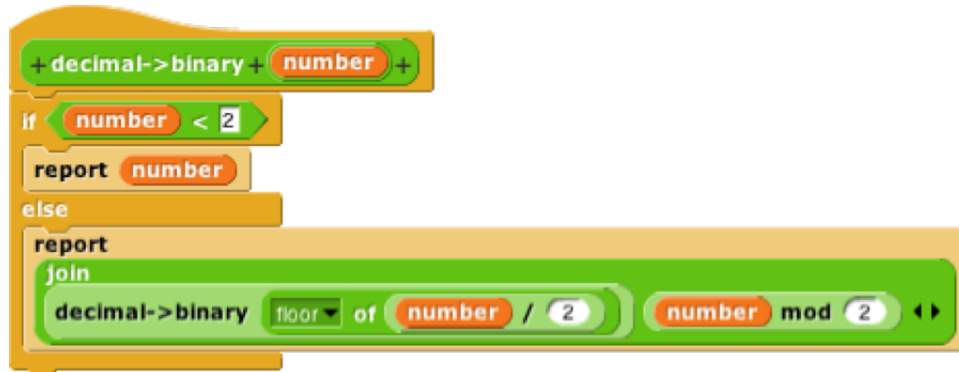
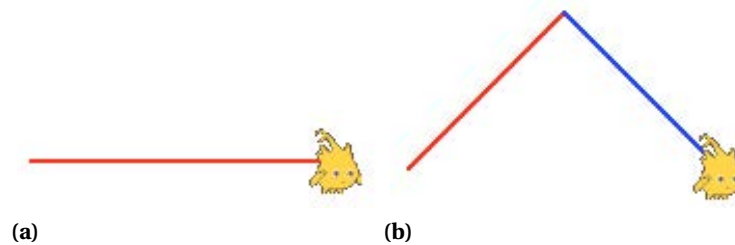


Figure 7.1 Base Snap! code for a Decimal to Binary conversion block.

This is a recursive function that must have an if-else structure that directly computes the smallest base case if number is the smallest acceptable value, and an else step that uses the code again for larger input numbers. For base 2, or binary, the base case is that the number fits in a single bit—that is, it has to be less than 2. If so, the number itself, 0 or 1, is the desired output, and the function reports back that number. See the if portion in Figure 7.1.

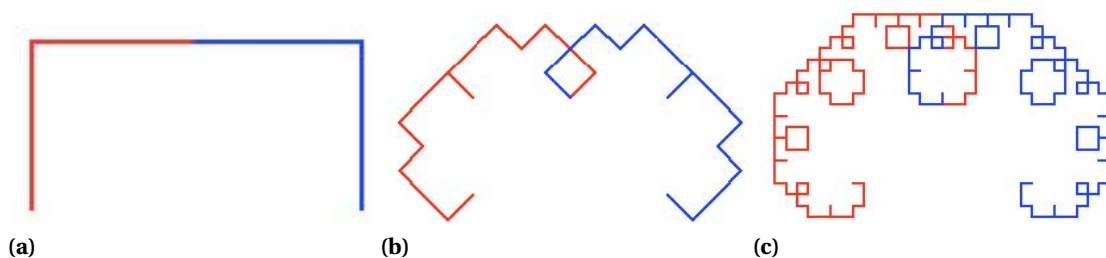
In the else portion of the code, known as the recursive step, the rightmost bit of the result is the remainder of dividing the number by 2. That is, even numbers end with 0, and odd numbers end with 1. The rest of the result is a recursive call on the (integer) quotient of the number divided by 2. The combiner is `join` because we want to string the digits together. It may be surprising that we don't use an arithmetic operator, since we're working with numbers, but the desired result is a numeral, which is a visible representation of a number, rather than the numeric value itself. A numeral is a text string, so the combiner is a string operation.

- **C-Curve** From the 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. Below the base case is that the sprite draws a single line. The sprite starts facing right and faces right at the end. (Hint - the direction that the sprite points at the end is important! It should point in the same direction it did at the beginning of the recursive call.)"



**Figure 7.2** Level 1 and 2 of the C-Curve algorithm

In the next level, start facing right and end facing right but repeat the previous level twice (red and blue below)."



**Figure 7.3** Extended recursion levels of the C-Curve

## 7.3 Results

A total of 19 participants were recruited with 9 teachers and 10 students. A total of 15 participants (8 teachers, 7 students) completed the Binary Conversion activity, and a total of 14 participants (6 teachers, 8 students) completed the C-Curve lab. Due to the flexibility of the 3-part online study, participants were able to stop and continue working at later times. For each part of the study, we removed data where participants spent less than 2 minutes on that part. This was because a duration of less than two minutes does not allow enough time for reading the question. The multiple breaking points coupled with time filters cause the differences in the number of participants for each lab. A breakdown of participation demographics is available in Table 7.1

**Table 7.1** Participant breakdown for the Spring 2017 study. Participants in the middle row are part of both data sets.

Under Review	Total # of Participants	Gender	Occupation	Education
Binary Converter Only	N=5	3 Female, 2 Male	3 CSP Teachers 2 Math Ed Majors	3 Bachelor degrees 1 Upperclassman 1 Underclassman
Binary & C-Curve	N=10	5 Female, 5 Male	4 CSP Teachers 1 Chem Teacher 3 Tech Ed Majors 2 Math Ed Majors	2 Master's of Education 3 Bachelor degrees 3 Upperclassmen 2 Underclassmen
C-Curve Only	N=4	3 Female, 1 Male	1 Business Teacher 3 Math Ed Majors	1 Bachelor degree 2 Upperclassmen 1 Underclassman

### 7.3.1 Intra-class correlations

When analyzing this data, the first aspect we investigated was 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 use 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 using the SPSS statistical package 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 [Cic94].

We next computed reliability for each lab separately. For each of the lab descriptions, we calculated the combined reliability of both participant groups, just teachers, and then just students. These results are described below.

#### **7.3.1.1 Binary Conversion Lab ICC**

**Teachers + Students** We calculated an overall level of reliability using the 15 total Binary participants. ICC estimates and their 95% confidence intervals were calculated based on a mean-rating ( $k = 15$ ), consistency-agreement, 2-way random-effects model. We calculate our results to be  $ICC = 0.90$  with 95% confidence interval = 0.80-0.96. Based on the ICC results, we concluded that the inter-rater reliability is “excellent.”

**Teachers ICC on Binary Lab** We calculated the level of reliability among teachers using the 8 corresponding Binary participants. ICC estimates and their 95% confidence intervals were calculated based on a mean-rating ( $k = 8$ ), consistency-agreement, 2-way random-effects model. We calculate our results to be  $ICC = 0.87$  with 95% confidence interval = 0.73-0.95. Based on the ICC results, we concluded that the inter-rater reliability is “good” to “excellent.”

**Students ICC on Binary Lab** We calculated the level of reliability among students using the 7 corresponding Binary participants. ICC estimates and their 95% confidence intervals were calculated based on a mean-rating ( $k = 7$ ), consistency-agreement, 2-way random-effects model. We calculate our results to be  $ICC = 0.74$  with 95% confidence interval = 0.50-0.89. Based on the ICC results, we concluded that the inter-rater reliability is “fair” to “excellent.”

#### **7.3.1.2 C-Curve Generation ICC**

**Teachers + Students** We calculated an overall level of reliability using the 14 total C-Curve participants. ICC estimates and their 95% confidence intervals were calculated based on a mean-rating ( $k = 14$ ), consistency-agreement, 2-way random-effects model. We calculate our results to be  $ICC = 0.76$  with 95% confidence interval = 0.57-0.90. Based on the ICC results, we concluded that the inter-rater reliability is “fair” to “excellent.”

**Teachers ICC on C-Curve** We calculated the level of reliability among teachers using the 6 corresponding C-Curve participants. ICC estimates and their 95% confidence intervals were calculated

based on a mean-rating ( $k = 6$ ), consistency-agreement, 2-way random-effects model. We calculate our results to be  $ICC = 0.74$  with 95% confidence interval = 0.50-0.89. Based on the ICC results, we concluded that the inter-rater reliability is “fair” to “excellent.”

**Students ICC on C-Curve** We calculated the level of reliability among students using the 8 corresponding C-Curve participants. ICC estimates and their 95% confidence intervals were calculated based on a mean-rating ( $k = 8$ ), consistency-agreement, 2-way random-effects model. We calculate our results to be  $ICC = 0.39$  with 95% confidence interval = -0.16-0.74. Based on the ICC results, we concluded that the inter-rater reliability is “poor” to “good.”

These results are discussed further below in Section 7.4.

### 7.3.2 Heat Mapping and Visualization Analysis

As part of the study, participants were asked to identify the code that influenced their decision for giving out each rating. For example, they might highlight a conditional statement when determining the amount of mathematical logic being used in the program. In total, participants made 1000+ indications, with 540 made by 15 participants who completed the Binary Conversion portion and 468 made by 13 participants who completed the C-Curve activity. One participant only completed the grading portion of C-Curve and not the code identification task.

Code identification data was hand-tagged using Gradescope [Gra14], a web app designed for grading paper-based assignments. This was useful as the post-survey identification answers were in PDF format. Using Gradescope, each of the code samples were set up as a different class assignment (Binary1, Binary2, etc.). We outlined each assignment in the system marking each relevant identification question. This made it possible to quickly focus on only the relevant code data for each portion of the rubric-graded responses. Each of the 6 assignments had a total of 6 questions, one for each category of the rubric.

Multiple researchers were able to work in parallel tagging all 36 grading questions. Researcher 1 tagged 1/3 of the data, and trained researcher 2 how to do so as well. After researcher 2 tagged 1/3 of the data, researcher 1 went back over and confirmed the tags. Researcher 2 then finished tagging the last 1/3 of the data. Tags were straightforward with terms such as 'header,' 'top if,' and 'join block.' The built-in statistics package would then display the frequencies of each tag for every question.

In the sections below, we present a brief formation of the rubric used for the lab assignments, followed by representative samples of code indication by participants.

### 7.3.2.1 Binary Conversion

Table 7.2 shows each of the categories presented on the rubric, the simple name followed by the learning objectives used to form the category (L) and the descriptive code that would appear in the rubric ratings (R). The rubric text provided is for the highest score, 4. For a full length version of this rubric please see the final Appendix.

**Table 7.2** A sample of the categories and learning goals for the Binary Conversion Rubric.

Category	Learning Goals (L) and Rubric Description (R)
Abstraction	L: An Abstraction generalizes functionality with input parameters that allow software reuse. R: Code properly separated into multiple abstractions. Should have separate blocks defined.
Visualization	L: Visualization tools and software can communicate information about data. R: Custom block reports a string of values the converts from decimal to base 3-10 (input value). 6 base 3 ->20
Mathematics	L: Numbers can be converted from any base to any other base. Iteration is the repetition of part of an algorithm until a condition is met or for a specified number of times. Numbers and numerical concepts are fundamental to programming. R: Code uses mathematics (floor, mod, etc) to calculate the converted numbers. Mathematics is used to automate function calls.
Parameters	L: Parameters provide different values as input to procedures when they are called in a program. Code blocks and abstract functions use parameters to increase usability and are easily changeable for testing. R: Parameters allow users to input a number that is decimal (base 10) and a base value (up to 10).
Style	L: Program style can affect the determination of program correctness. Duplicated code can make it harder to reason about a program. R: Code cleanly organized and laid out to be read easily. Abstractions/blocks are properly composed.
Naming	L: Meaningful names for variables and procedures help people better understand programs. Documentation about program components, such as blocks and procedures, helps in developing and maintaining programs. R: Naming of functions, including abstractions, accurately represents the problem being solved.

The first code sample shown in Figure 7.4 represents the Snap! code for the first Binary Conversion solution participants were given. This code sample represents high-performing student code.

---

**Algorithm 1 Binary Converter (High)**

---

```

1: Procedure decimal to base base
2: if decimal < 1 or decimal = 1 then
3:   report 1
4: else
5:   if decimal mod base > 10 or decimal mod base = 10 then
6:     report join((floor(decimal/base) to base base),
7:       unicode decimal mod base +87 as letter)
8:   else
9:     report join((floor(decimal/base) to base base), decimal mod base)
10:  end if
10: end if

```

---

**Figure 7.4** Heat map of areas that influenced grader decision for ‘Mathematics’ in Binary Conversion 1

Areas selected by participants are shaded, and colored according to whether the rubric category is actually relevant to the code. Areas highlighted in red are misdirected, and areas in green are highly relevant to the CT concept being assessed. Areas highlighted in Yellow are not incorrect, but are not the most important to the category. The intensity of the color correlates to the frequency of its selection by participants. In Figure 7.4 the intensity ranges from 7% in the red highlighting of the parameters in the procedure heading, to 67% in green of the math operators on line 6.

---

**Algorithm 2 Binary Converter (Low)**

---

```

1: Procedure decimal to base base
2: if decimal < base then
3:   report decimal
4: else
5:   report join “hello ”, “world”
6: end if

```

Extra code blocks:

```

___ mod base
floor of 10
decimal

```

---

**Figure 7.5** Heat map of areas that influenced grader decision for ‘Mathematics’ in Binary Conversion 2

The second code sample shown in Figure 7.5 represents the Snap! code for the second Binary Conversion solution participants were given. This code sample represents lower performing student code. In Figure 7.5 the intensity ranges from 7% green highlighting on line 5 to 40% highlight of floor(10). The extra code blocks are not a part of the main code block, and do not function when the program runs. As a reminder, participants were given the code in a live environment to test out and explore in detail. The large section of red highlighting in Figure 7.5 represents the 15% of participants who selected no code for this assignment review. This is the only sample where participants selected no code to accompany their scores.

The final code sample shown in Figure 7.6 represents the Snap! code for the third Binary Conversion solution participants were given. This code sample represents typical-performing student code. In Figure 7.6 the intensity ranges from 15% highlighting of the procedure declaration on line 1, to 53% highlighting of floor(number/base) on line 5.

---

### 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

```

---

Figure 7.6 Heat map of areas that influenced grader decision for ‘Mathematics’ in Binary Conversion 3

#### 7.3.2.2 C-Curve

Table 7.3 shows each of the categories presented on the C-Curve rubric, the simple name followed by the learning objectives used to form the category (L) and the descriptive code that would appear in the rubric ratings (R). The rubric text provided is for the highest score, 4.

The first code sample shown in Figure 7.7 represents the Snap! code for the first C-Curve solution participants were given. This code sample represents typical performing student code. The intensity of the color correlates to the frequency in which it has been selected. In Figure 7.7 the intensity ranges from 15% in the yellow highlighting of the whole program to 76% in green of *size*  $\sqrt{(2)}=2$

**Table 7.3** A sample of the categories and learning goals for the C-Curve Rubric.

Category	Learning Goals (L) and Rubric Description (R)
Abstraction	L: An Abstraction generalizes functionality with input parameters that allow software reuse. R: Code properly separated into multiple abstractions. There should be a hierarchy of functions (the main C-Curve fractal level should call other functions).
Visualization	L: Visualization tools and software can communicate information about data. R: C-Curve is styled correctly (see stage6 in the handout). There exists a variable size and a variable number of levels. Levels are correctly added to the corners of the main center C-Curve (angles all correct)
Mathematics	L: Algorithms can be combined to make new algorithms. Using existing correct algorithms as building blocks for constructing a new algorithm helps ensure the new algorithm is correct. Numbers and numerical concepts are fundamental to programming. R: Code uses mathematics (loops, operations, ...) to monitor length of C-Curve, how many levels of C-Curve to make.
Parameters	L: Parameters provide different values as input to procedures when they are called in a program. Code blocks and abstract functions use parameters to increase usability and are easily changeable for testing. R: Parameters allow users to input size of C-Curve and number of levels.
Style	L: Program style can affect the determination of program correctness. Duplicated code can make it harder to reason about a program. R: Code cleanly organized and laid out to be read easily. Abstractions/blocks are properly composed.
Naming	L: Meaningful names for variables and procedures help people better understand programs. Documentation about program components, such as blocks and procedures, helps in developing and maintaining programs. R: Naming of functions, including abstractions, accurately represents the problem being solved.

on line 6 and 8.

The second code sample shown in Figure 7.8 represents the Snap! code for the second C-Curve solution participants were given. This code sample represents high performing student code. The intensity of the color correlates to the frequency in which it has been selected. In Figure 7.8 the intensity ranges from 7% in the yellow highlighting of the whole program to 76% in green highlighting of math operators on lines 6 through 8.

---

**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 7.7 Heat map of areas that influenced grader decision for 'Mathematics' in C-Curve Sample 1

---

**Algorithm 2 C-Curve (high)**

---

```
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) and level (level - 1)
7:   turn clockwise ((level × -45) + 180)°
8:   ccurve size: (size ÷ √2) and level (level - 1)
9: end if
```

---

Figure 7.8 Heat map of areas that influenced grader decision for 'Mathematics' in C-Curve Sample 2

The third code sample shown in Figure 7.9 represents the Snap! code for the third C-Curve solution participants were given. This code sample represents low-performing student code because of its repetitive nature and incorrect solution. In Figure 7.9 the selection frequency (intensity) ranges from 15% in the yellow highlighting of the whole program to 53% in green of  $(size=2)(level - 1)$  on line 5.

---

**Algorithm 3 C-Curve (Low)**

---

```
1: Procedure ccurve size level
2: if level = 1 then
3:   move size steps
4: else
5:   ccurve (size/2) (level - 1)
6:   turn counterclockwise 45°
7:   ccurve (size/2) (level - 1)
8:   turn clockwise 45°
9:   ccurve (size/2) (level - 1)
10:  turn counterclockwise 45°
11:  ccurve (size/2) (level - 1)
12:  turn clockwise 45°
13: end if
```

---

**Figure 7.9** Heat map of areas that influenced grader decision for ‘Mathematics’ in C-Curve Sample 3

## 7.4 Discussion

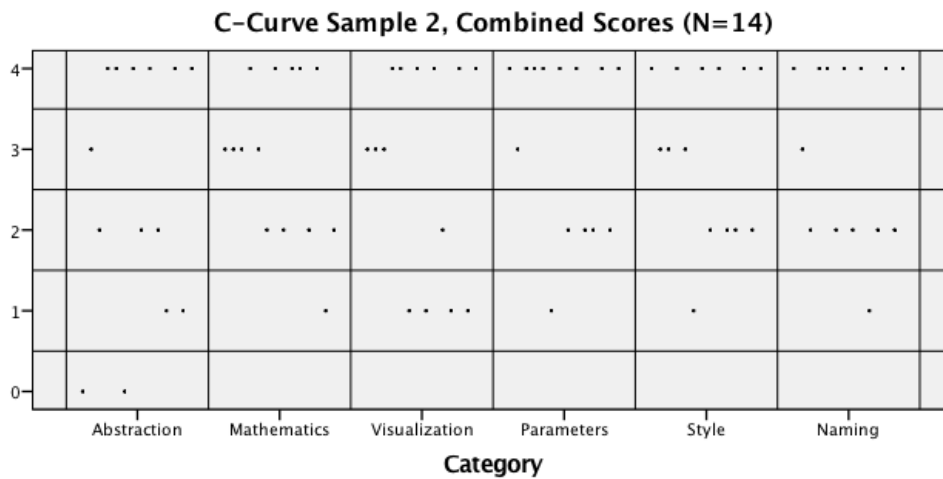
When analyzing this data, the first aspect we investigated was the consistency model for inter-rater reliability between raters. To determine these values we analyzed average consistency across both projects with an ICC value of .74 which is rated acceptable for research purposes<sup>2</sup>. This is promising as we have mixed levels of coding samples and CS Principles teaching experience (0-2 years). When broken down by project, we notice there is a discrepancy between experienced and inexperienced teachers.

For the Binary Conversion project, which is the easier of our two projects, we find that CS Principles teachers scored very high with an ICC of .87. The STEM education students also scored acceptably high with .74. Both of these statistics show that our graders are giving students consistent grades while using our rubrics, and for experienced CS Principles teachers, their level of agreement is even tighter. This is expected for a simple program like the Binary Conversion where students only need to add an additional variable to solve the problem given the starter code.

Although both programs use recursion, the C-Curve program is noticeably harder as students must identify the recursive pattern on their own before implementing the problem in code. The overall ICC for the combined participant group is .76, for teachers the score is close with .74. However, STEM Education students as a group only scored an ICC of .38. Although the ICC score is across all three student samples, this difference in correlation can be visibly seen in the distributions for

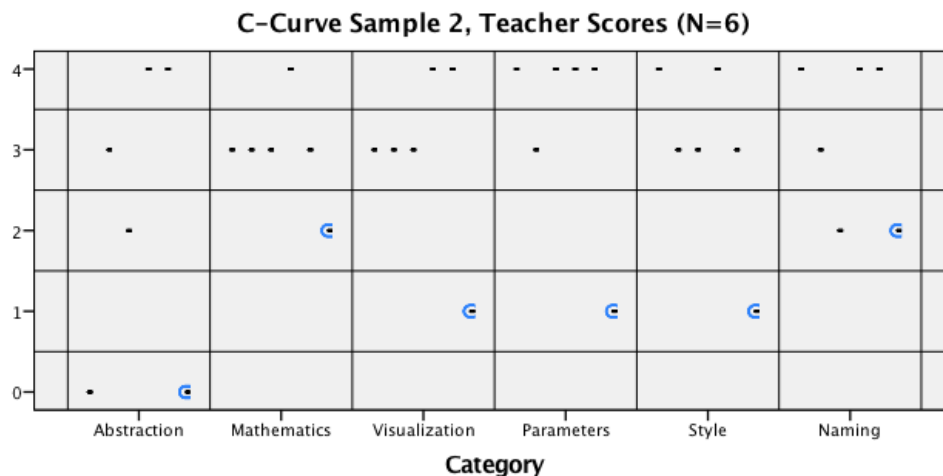
---

<sup>2</sup>For high stakes clinical research a value of .90 is considered sufficient [GM09]

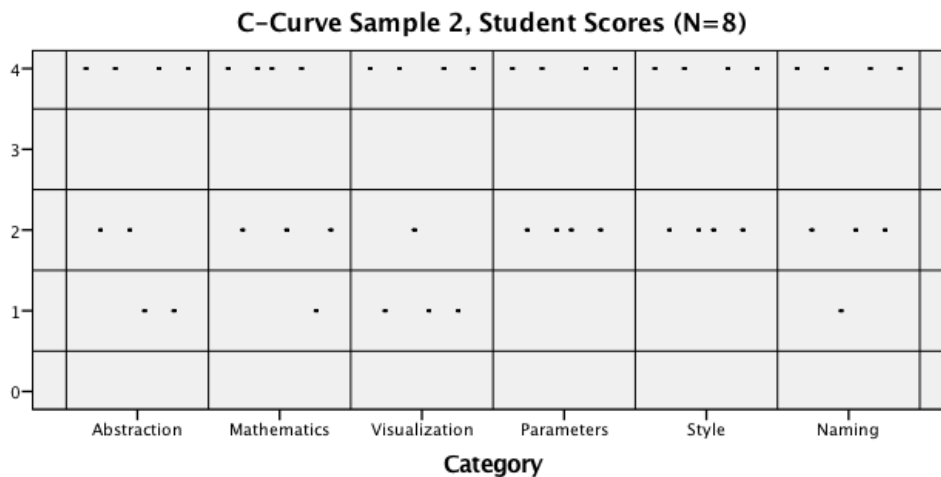


**Figure 7.10** Combined score distribution for C-Cure Sample 2, ICC(2,14)=.75.

C-Curve sample 2. Figure 7.10 shows the score distribution for sample 2 of the C-Curve assignment. The ICC value reflects the density of score distribution. When looking at just the teacher data in Figure 7.11, there is a similar density to the overall scores. The student score distribution in Figure 7.12 is strikingly different. Figure 7.12 shows how the scores for the students were distributed in a very polarized fashion.



**Figure 7.11** Teacher score distribution for C-Cure Sample 2, ICC(2, 6)=.74.



**Figure 7.12** Student score distribution for C-Curve Sample 2, (ICC2, 8)=.31.

When investigating the lower correlations for C-Curve compared to Binary, we were able to identify several key findings. To demonstrate these, we use C-Curve sample 2 as an example. The first finding we identified was the polarization of student scores; students either gave out high scores of 4, or low numbers between 1-2. The appropriate score set for Sample 2, our high performing student sample, is straight 4s, which our CS Principles teachers were better able to identify.

To understand the reason for these low scores, we split our students by major and found that both of the Technology education students gave consistently lower scores compared to Math education students. We later went back to our teacher data to find a similar trend based on experience. The lowest scoring teacher, P6, highlighted in blue circles in Figure 7.11, was our Chemistry teacher who hadn't taught CS Principles yet. We believe that as Math education majors tend to take more math classes and have seen recursive functions or fractals before, they were more familiar and thus comfortable with grading the C-Curve samples. This could be echoed in the CS Principles data as 4 of the other CS Principles teachers selected Math as their primary or secondary teaching field and 1 teacher listed Business. The remaining two students who gave lower scores were identifiable as participants with noticeably shorter duration periods when completing the post survey. This implies that they did not consider the problem fully and moved on quickly. The polarization effect we see in C-Curve sample 2 is also present in samples 1 and 3, although not as severely.

The next aspect that we looked at for analysis is the code selection process. Figures 7.4-7.9 demonstrate selections made by participants. From these examples we see that participants were better able to match computational thinking concepts from rubric to code when code samples

resembled mid to high level student performance. As student code deviated from the intended solutions, participants diversified where they were looking to evaluate the program as seen in Figure 7.5 and Figure 7.9. In Figure 7.5 the desired code elements are not present leading some participants to select no code, where as others will select unused code. Selecting unused code could be a reflection of wanting to give partial credit for student work. In Figure 7.5 on the other hand, the intensity levels are lower, meaning participants are more varied in where they are looking. It is interesting to note, that in Figure 7.5 there is a higher intensity on `turn clockwise 45` than on the other turns and recursive calls. This indicates that several of the participants were able to identify not just faults in conditional logic, but also in programming logic. Either through running the simulation or comparing against the other working student samples, participants were able to determine that the `turn clockwise` call on line 8 was incorrect.

Frequency tables for code selection across samples have shown that graders are able to locate parameters, variables and mathematical concepts consistently. However, they less frequently tag the logical operators which also make up the learning objectives in the mathematics category. It may be that participants were using just their own understanding of mathematics to apply the rubrics, rather than carefully reviewing the written descriptions of what was expected at each level. To help participants better identify appropriate code, I will revise the short labels to replace math and logic with calculations and comparisons.

When it came to identifying abstraction, the majority of participants selected appropriate areas such as the block declaration, which represents the abstraction of determining a single function and defining what it should do, or 'Everything below' meaning they looked at the overall program as a cohesive whole opposed to focusing on certain points. We believe this indicates that participants were able to understand that abstraction refers to the big picture of the programs functioning parts.

Furthermore, participants were able to consistently rank 'Style' with little trouble. We believe this is due to the non-technical nature of the category and the teachers' ability to determine organizational qualities within student code. We do find however, that the 'Naming' category, which is similar to style in its less technical nature, had consistently more diverse responses. While the style category is concerned with efficiency and minimizing duplicate code, naming is about understanding what variables and functions are intended to do and naming them appropriately. The diversity in code selection indicates that for harder problems such as the C-Curve, teachers need to have an understanding of the programming elements and their outcomes to answer such categories. This could be supported through companion resources such as detailed assignment walk-throughs or annotated solutions.

## 7.5 Conclusions

We surveyed 19 participants ranging from CS Principles teachers with 2 years of experience teaching the course to STEM Education undergrads who are still in the pipeline. We specifically targeted novice CS Principles teachers to assess their use of our rubrics and how well they could identify the computational thinking associated with lab assignments in student code. In total, participants tagged over 1000 lines of code relating to their understanding of how CT appears in programs.

In analyzing this data, we first examined inter-rater reliability through intra-class coefficient consistency (ICC). Overall, we found participants to have a sufficient agreement level  $>.76$ . This is consistent with the inter-rater reliability levels achieved with the original coding samples and rubrics created from the Delphi method in Chapter 4. We found that on the simpler Binary Conversion problem consistency improved for both student and teacher groups. On the more complex C-Curve assignment, teachers performed consistently, but the less experienced students had very polarized results, with the Math education students aligning better with the true scores.

The second facet of analysis was heat map frequency analysis of the code selection data. Data showed that as code samples deviated from efficient correct solutions, participants had a harder time identifying the relevant aspects of code. This suggests that the rubrics could use more support for identifying CT in lower quality code samples, such as low level block descriptions. In terms of being able to identify types of CT, evidence suggests that participants were able to consistently identify regions of abstraction, mathematical operations, parameters, and code styling. Participants less frequently indicated conditional logic, which is an equal part of the mathematics category. This inclines us to consider revising the short category titles to be more obviously include the respective learning objectives, for example using comparisons and calculations instead of logic and math. Participants also had more diverse and diluted frequencies than the other categories in their selections for the the naming rubric category, which requires graders to identify the intended functionality of custom code blocks and variables. This suggests that new CS Principles teachers could benefit from companion resources such as annotated solution files that would more specifically demonstrate samples that do and particularly that do not mean the desired learning outcomes.

The rubrics used in this study were created using a modified NGT model outlined in Chapter 6 to quickly produce a full suite of rubrics for the BJC CS Principles course. Given the unprecedented growth in incoming CS Principles students and teachers, we need to produce a high quantity of materials in a short time. However, when rapidly producing these materials we want to maintain a sufficient level of quality and standards. This study demonstrates that active CS Principles teachers are able to use these rubrics to assess student code consistently and accurately. This study also sheds light on the specific support needs of new CS Principles teachers with diverse backgrounds.

## CHAPTER

# 8

# RUBRIC DEVELOPMENT ASSESSMENT USING WOCA

## 8.1 Introduction

In previous studies, we attempted to create well-defined rubrics for a Computer Science Principles course. Our approach has evolved from task-based rubrics founded in auto-tutor assessment strategies to learning objective-oriented rubrics built by group decision techniques such as the Delphi method and the Nominal Group Technique (NGT). Although our previous research has found initial inter-rater reliability through Cohen's Kappa, we have not yet focused on validity. Despite the gravitation in research towards Messick's six-part validity, we have decided that his approach is not fit for our rubric validation efforts [Mes96]. Through a field survey of validity in assessment, we found that instead, we should focus on measuring validity and other quality criteria through Baartman's Wheel of Competency Assessment (WoCA)[Baa06]. When measuring our rubrics against Baartman's Wheel, we argue that we have met 11 out of 12 of the competencies, thus supporting the validity of our rubrics.

The rest of this paper will discuss options for validity including both Messick's and Baartman's methods. It will then explore the ways in which our rubrics and their creation process meet the

qualifications listed out in WoCA.

## **8.2 Background**

As budgetary restrictions and reform in public education moves to the forefront of political agendas, leaders and decision makers want proof that the strategies being taken are having positive effects on students and learning. Therefore, there is a strong push for high quality assessments. Rubrics, a simple form of assessment, also have to be of high quality to have a positive effect in the classroom. During the 2000 Annual Meeting of the American Educational Research Association, Arter suggested four traits for quality: content, clarity, practicality, and technical soundness [Art00].

Technical soundness refers to the reliability and validity of scoring rubrics and other assessment items. In the context of this section, we define reliability as the ability for a student's work to repeatedly score the same marks on the same rubric. Furthermore, we define validity to refer to how well the assessment measures what it is supposed to measure. In the next subsections we present a discussion around reliability and validity and how they are used and measured by other researchers.

### **8.2.1 Reliability**

Most researchers agree when it comes to terms and definitions for general reliability as it relates to educational assessments [JS07; Ste04]. Reliability is separated out into two main concepts: 1) inter-rater reliability, how well two different raters match on assessing student assignments and 2) intra-rater reliability, how well a single rater will agree with their previous marks for matching work.

Inter-rater reliability is typically reported in three different ways: through consistency estimates, consensus agreements, and measurement estimates. Consistency estimates refer to the relation where two raters grade similarly, but perhaps with differing judgment levels. For example, an A for rater 1 is a B for rater 2, but as a whole each student assignment has the same relationship to each other (e.g. the same rank order). These estimates are often presented through correlation coefficients. As aggregated by Jonsson, Pearson's correlation is the most commonly reported, although many researchers do not specify which correlation they are using. These correlations are typically between .55 and .75, meaning that most researchers fail to meet the .70 acceptability criterion [JS07].

The second way in which inter-rater reliability is reported is through consensus agreement. Consensus agreement is frequently reported due to its low barrier for calculation (percent agreement and Cohen's kappa). Consensus agreement directly measures how two raters match when grading. As Jonsson's findings show, researchers report exact agreement scores below 70% and adjacency or off-by-one agreements greater than 90%, again showing that most researchers fail to hit the 70%

exact agreement mark, but do display a consistency in ratings. Jonsson notes that as the levels in the rubric decrease, it's more likely to achieve consensus by chance– and this should be accounted for in the results.

The final reporter for inter-rater reliability is measurement estimate. Measurement estimates try to preserve as much information as possible from the judges and incorporate it into an inter-rater reliability model [Ste04]. The many-facets Rasch model determines the 'severeness' of a rater and shows the spread across all judges. Additionally, this model gives an Infit statistic that indicates the level of unpredictable variation in responses. These measurement estimates usually require special software to calculate and do not handle nominal data. Other variations for implementation are generalizability theory and principle components analysis. Due to the cumbersome nature of measurement estimates, they are often ignored and not reported [JS07]

Intra-rater reliability statistics are also frequently ignored. Intra-rater reliability often requires a test-retest approach, and with well-defined rubric categories is not a major concern to many researchers. In situations where intra-rater reliability becomes an issue and users relay difficulties, the rubric authors will usually revise the category descriptions to be less ambiguous.

Overall, reliability is very important to high-stakes and national assessments that are graded by many different raters, however, reliability is not as crucial in low-stakes classroom based assessments. In typical classroom settings, a single teacher will grade all assignments by using an analytical rubric that compartmentalizes each of the assessed features. For those desiring to meet reliability standards ( $k > .7$ ), two raters, under certain conditions, are enough to produce acceptable levels for inter-rater agreement [Bak96]. Additionally, research supports that reliability can be increased by using topic-specific rubrics, especially if teachers have been trained how to use them.

### **8.2.2 Validity**

There are different views with respect to validity when it comes to assessments [ML00; Mes96]. In the classical model there are three main forms of validity: Content Validity, Criterion Validity, and Construct Validity. Content validity is a type of validity that examines the test content to determine whether it covers a representative sample of the domain to be measured. Criterion validity is the extent to which a measure is related to an outcome. Finally, construct validity refers to the degree to which a test measures what it claims, or purports, to be measuring.

There is a newer model however, which frames validity as one unit, Messick's unified theory of Construct Validity. In this model of validity there are six different aspects that are all connected and that depend on the quality of the construct [Mes96]. AERA adopted this model in 1999 and worked with several other organizations to frame the *Standards for Educational and Psychological Testing*

around them [Ass99]. These standards are the benchmark for national and high stakes testing. A description of Messick's six aspects are listed below, although they are not necessarily intuitive to those outside the testing field.

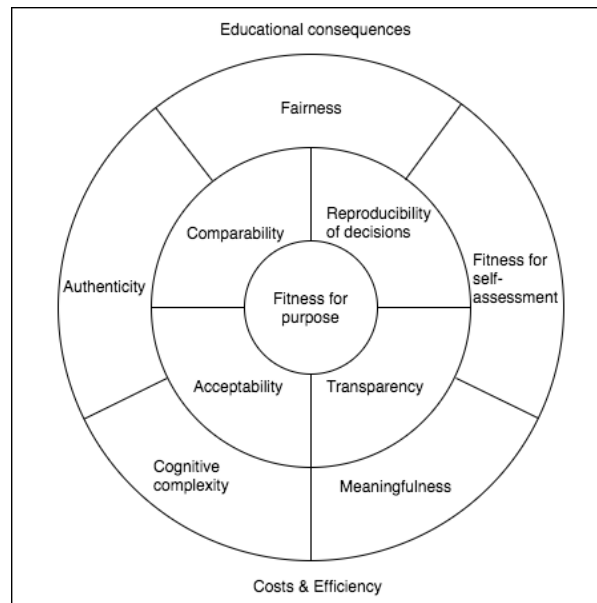
- *Content* This aspect of construct validity includes evidence of content relevance, representativeness, and technical quality [Len56; Mes89].
- *Substance* The substance aspect refers to theoretical rationales for the observed consistencies in test responses, including process models of task performance [Emb83], along with empirical evidence that the theoretical processes are actually engaged by respondents in the assessment tasks.
- *Structure* The structure aspect appraises the fidelity of the scoring structure to the structure of the construct domain [Loe57].
- *External Variables* This aspect of construct validity includes convergent and discriminant evidence from multitrait-multimethod comparisons [CF59], as well as evidence of criterion relevance and applied utility [CG65].
- *Generalizability* The generalizability aspect examines the extent to which score properties and interpretations generalize to and across population groups, settings, and tasks [TDC79; Shu70], including validity generalization of test-criterion relationships [SH14].
- *Consequences* This aspect appraises the value implications of score interpretation as a basis for action as well as the actual and potential consequences of test use, especially in regard to sources of invalidity related to issues of bias, fairness, and distributive justice [Mes80; Mes89].

Both the classical model of validity and the unified concept of construct validity are useful to the users of the assessment information when it comes to high-stakes testing. However, the frameworks cause practical implementation problems as it is often not conceptually clear what the authors are looking for in terms of evidence, especially to those outside of educational psychology and psychometrics. A large part of the discrepancy for our needs is due to the nature of scoring rubrics for problem-based assignments; they lack the necessary testing capabilities to carry out recommended empirical procedures and psychometric statistics such as national survey items, post-course evaluations, and factor analysis. [JS07].

The scoring rubrics that we have developed are intended for teachers to gauge whether or not their students are meeting the necessary learning goals of the individual lab assignments and if needed, adjust their instruction. Completing full stack construct validity testing is impractical for such a low-stakes formative assessment.

### 8.2.3 Low-Stakes Problem-Based Assessments

As Baartman reasons, measures of reliability or validity are not fundamentally wrong for [mixed assessment]<sup>1</sup>, but they should be applied in a different way and be combined with other quality criteria that are especially important for competency assessment. We argue that this applies to our assessments as well, since they deviate from the classical tests in which the classical methods were designed.



**Figure 8.1** Baartman's Adapted Framework: The Wheel of Competency Assessment [Baa06]

Baartman presents a case for the Wheel of Competency Agreement, as shown in Figure 8.1. This validated framework suggests 12 criterion for quality assessment of competency assessment [Baa06]. Neither validity nor reliability are explicitly listed as they present larger container concepts that are often misconstrued. Instead this wheel is based on singular concepts decoupled from larger complicated processes making it more accessible to outside researchers. Baartman maintains the inter-connectedness of the concepts through the circular shape of the wheel, but gives each one its own zone. Each of these 12 concepts is listed and described briefly below.

- *Fitness for purpose*: Assessment should align with the goal of the learning process and with

<sup>1</sup>This article argues for integrating different assessment methods into a Competency Assessment Program, in which newer forms of assessment can be used in combination with more classical methods.

the instruction given.

- *Comparability*: Scoring should occur in a consistent way using the same criteria for all learners.
- *Acceptability*: The assessment has to be accepted by those in the profession. Acceptability has to do with the attitudes and views of the stakeholders.
- *Transparency*: Is the assessment clear and understandable to all participants? A possible indication is to check whether learners can judge themselves and other learners as accurately as trained assessors.
- *Reproducibility of decisions*: Decisions about the learner are made accurately and do not depend on the assessors or the specific assessment situation. It is not necessary for decisions to be objective to be reproducible.
- *Authenticity*: relates to the degree of resemblance of the assessment to the future profession life. Does the assessment evaluate competencies needed in the future workplace?
- *Fairness*: The assessment should not show bias towards a certain group of learners and should only reflect the knowledge, skills and attitudes of the competency at stake. It should not depend on unfamiliar and/or unrelated cultural aspects.
- *Cognitive Complexity*: Assessment tasks should reflect the presence of higher cognitive skills and elicit the thinking process used by experts to solve complex problems in their fields.
- *Meaningfulness*: The assessment should have a significant value for both teachers and learners. A possible way to increase meaningfulness is to include learners in the development of the assessment process.
- *Fitness for Self-Assessment*: The assessment assists in self-regulated learning by making clear what the criterion are, by showing weaknesses and by stimulating reflection on the learning process.
- *Educational Consequences*: How do the intended and unintended, positive and negative effects of the assessment cause teachers and learners to view the goals of education and adjust their learning activities accordingly? e.g. washback;<sup>2</sup>.
- *Costs & Efficiencies*: The time and resources needed to develop and carry out the assessment are justified by the positive effects, such as improvements in learning and teaching.

---

<sup>2</sup>A prevailing phenomena in education. Where "what is assessed becomes what is valued, which becomes what is taught" [CC04].

## **8.3 Measuring Criteria for Assessment**

### **8.3.1 Why Rigorously Evaluate Rubrics**

As the previous section demonstrates, there are different viewpoints when it comes to quality assessment standards. Arter asserts that the validation process should be reserved for situations in which quality criteria for important learning targets are somewhat fuzzy and can benefit from a comprehensive approach [Art00]. Such a heavyweight process might seem overkill for simple high school scoring rubrics, however, the number of new K-12 computer science teachers with non-computing backgrounds is growing exponentially. With such unprecedented growth, comes vastly different levels of teacher preparedness.

If we hope to get widespread adoption of newly formed K-12 computing courses throughout the United States, including in rural areas further away from university hubs, we need well-complimented courses with rich support materials so that active teachers can pick and choose how to make the classes theirs. Ni's 2009 study showed that for teachers who adopted computing curricula, over 50% modified existing courses to use some of their content; only 7% used the full complete curriculum [Ni09]. A national standard of fully vetted modules gives more options for teachers to pick and choose assignments for blended courses. When teachers feel a sense of ownership over their course curricula, they are more likely to persist in teaching the course and to workshop with others to continually improve their course for their own specific needs and students [Ni09].

### **8.3.2 How to Evaluate Rubrics Meaningfully**

In the classical sense of educational testing, reliability and validity are key players. Messick worked with the AERA and other educational stakeholders to integrate his new construct validity into national assessment standards for student education. Although these standards are important for the high-stakes testing that came out of the No Child Left Behind era [Bus01], these standards are difficult to apply to new forms of problem-based assessments with open-ended solutions.

The psychometric standards of high-stakes testing do not meet the cost-effective needs and utility of rubrics. The primary goals of the rubrics we are developing are friendliness and clarity for new CS teachers. These low-stakes assessments are designed as building blocks for student learning and should be evaluated as such. The flexibility of the Wheel of Competency Assessment allows us to evaluate certain aspects of reliability and validity while still maintaining a broader, more practical scope for scoring rubric evaluation.

## 8.4 Aligning Research to the Wheel of Competency Assessment

In a previous study [Cat18], we created a hybrid process to quickly create rubrics for the Beauty & Joy of Computing (BJC), a variation of AP Computer Science Principles. In this analysis, we test our hybrid rubrics against the Wheel of Competency Assessment to estimate a measure of quality for our rubrics. There are 12 possible quality metrics; we however, will only be looking at 11 of them. As the scope of this research has focused primarily on teachers, we have not explored student use of the rubrics for *fitness for self-assessment* and will be leaving that research for future work. A summary of the evidence type and participants for each criteria is visible in Table 8.1. The remainder of this section provides descriptions for the evidence as to how our system of rubrics meets the 11 other quality criteria as outlined by Baartman. These descriptions are recapped in Tables 8.2, 8.3, and 8.4.

**Table 8.1** A summary of processes for meeting Wheel of Competency Criteria

	Criteria	Evidence Type	Participants
1	Fitness for Purpose	Delphi (Chapter 5)	Master Teachers
2	Comparability	Analytic Rubrics	N/A
3	Acceptability	2017 User Study	BJC Professional Development Attendees
4	Transparency	Chapter 5	Master Teachers and novice undergrads
5	Reproducibility of Decisions	Chapters 4 and 5	Master Teachers and novice undergrads
6	Authenticity	Delphi (Chapter 5)	SIGCSE Members
7	Fairness	Chapter 4	Novice BJC Teacher, Undergrad Experts
8	Cognitive Complexity	Chapter 4	Novice BJC Teacher, Undergrad Experts
9	Meaningfulness	Literature Review	N/A
10	Fitness for Self-Assessment	Not Tested	N/A
11	Costs & Efficiencies	Chapter 6	undergrads
12	Educational Consequences	Literature Review	N/A

*Fitness for purpose* The rubrics we developed were created using a systematic process [Cat18] guided by initial findings in [CB17]. During these processes we solicited CS Principles experts to identify the best fitting and most important learning objectives for each lab activity. These learning objectives were derived directly from the AP CS Principles Course Framework [Col17] on which each

CS Principles course is based. By using group decision techniques, we have reasonably identified the most appropriate learning targets to be used in each rubric demonstrating their fitness for purpose.

*Comparability* In the original context of Comparability, as described by Uhlenbeck, beginner teachers were being assessed on their ability to teach English as a Foreign Language (EFL) [Uhl02]. In that context, Uhlenbeck suggested guidelines for consistency of scoring portfolios, written exams, and simulations without necessarily using rubrics. Uhlenbeck suggested Candidates' responses should be scored consistently and according to the same criteria. Scoring should proceed systematically and the steps assessors take to reach a judgment should be open to inspection by others [Uhl02]. Each of these questions regarding comparability are addressed with the specification of the lab activities and clear performance criteria are described in each of the analytical rubrics.

*Acceptability* After introducing our suite of rubrics to Beauty and Joy of Computing teachers, we found primarily positive feedback. The only negative feedback received during any pilots pertained to minor spelling errors or inconsistent punctuation. When given the rubrics, teachers immediately felt comfortable grading with them. This was demonstrated during a series of summer 2017 BJC professional development workshops where teachers graded and submitted a total of 88 eCard and 99 Shopping List programming projects using our rubrics. Due to the lack of negative feedback and the ability of nearly 100 new BJC teachers to use the rubrics, we argue that our rubrics have been accepted by the relevant community.

*Transparency* Pairing well with Acceptability is Transparency, which relates to the clarity and understandability of the rubrics to participants. Hambleton suggests a good indication of transparency is "to check whether learners can judge themselves and other learners as accurately as trained assessors" [Ham96]. In our Spring 2016 study, we compared grading by an expert to that done by a novice learner. We compared grading abilities using both these newly formed learning-based rubrics, and previously developed task-based rubrics. We found that there was significantly greater accuracy in grading when using the learning-based rubrics. In addition to running the pilots with our summer 2017 PD teachers, the Spring 2016 study provides evidence for the clarity and understandability of the rubrics.

*Reproducibility of decisions* Reproducibility has been a primary focus of our past studies [Cat16; CB17]. This criteria does not require judgments to be objective, however we believe our scores generally are. In both Cateté 2016 and Cateté 2017 we have shown inter-rater reliability kappa values over 0.7. The statistics show that between experienced graders (kappa .79), as well as between novice and experienced graders (kappa .78) projects receive consistent scores. Furthermore, our Spring 2017 study with active CS Principles teachers in their first two years of teaching a computing course showed inter-correlation agreements of .74 for difficult assignments up to .87 for standard difficulty assignments. The consistency of scoring across time and graders indicates reproducibility of the

**Table 8.2** Evidence for Wheel of Competency Assessment pt. I: Term, Definition, Evidence

Fitness for Purpose	Comparability	Acceptability	Transparency
Assessment should align with the goal of the learning process and with the instruction given	Scoring should occur in a consistent way using the same criteria for all learners	Assessment has to be accepted by those in the profession (stakeholders)	Assessment is clear and understandable to all participants (learners can judge as well as trained assessors)
Rubrics designed around labs, CSP framework, and CT	Structure of analytic rubric provides consistent criteria	Multiply vetted and used by CSP teachers; Adopted by BJC EdX team	Shown in Study 3 comparing expert and novice graders

scoring decisions made based on the rubrics.

*Authenticity* In addition to the authenticity of the assessment criteria, there are four other dimensions of authenticity that relate to the process of completing the task: the assessment task, the physical context, the social context, and the assessment form [Gul04]. The assessment task for each lab is to ‘complete a programming task, turn in the final program to be graded,’ and the assessment form is our learning-based rubric. The physical and social contexts vary across class implementations. Some students take the course online, while others complete labs in pairs. This variety across classroom implementations echoes the variance in professional work environments. We argue that the rubrics are authentic because the assessment tasks (labs) were designed by CS professors, and the assessment forms (rubrics) were designed an expert panel of those familiar with the CS Principles course, and the computer science in universities and in industry. Since the chosen assessment criteria echo the needed competencies for future CS education and careers, they meet the authenticity criterion.

*Fairness* When examining the criterion of Fairness, we seek to minimize bias shown towards certain groups of learners and to reflect the knowledge, skills, and attitudes of the competencies at stake. We’ve vetted the rubrics for comprehension of category descriptions with HS teachers, university professors, and novice computing students. In Chapter 5, we have found that both high school honors computing students and non-computing majors in college receive similar grades for similarly demonstrated evidence of CT. Additionally, we found that by using evidence of CT learning objectives instead of task-based outcomes, students were equally judged on the beginner knowledge giving novice programmer students the chance to be successful without having to be masters at efficient programming. The rubrics themselves use learning objectives as the basis for

**Table 8.3** Evidence for Wheel of Competency Assessment pt. II: Term, Definition, Evidence

Reproducibility of Decisions	Authenticity	Fairness	Cognitive Complexity
Decisions about the learner are made accurately and do not depend on the assessors or situation	The assessment evaluates competencies needed in the future workplace	The assessment should only reflect the knowledge, skills & attitudes of the competency at stake	Assessment tasks should reflect the presence of higher cognitive skills to solve complex problems in field
Study 2, 3, and 5 address reliability of rubric across grader types	Group decision techniques in Study 3 and 4 vetted both importance and relatedness of the Learning Objectives	Study 2, auto-grader vs. learning-based rubrics, shows drop in bias towards expert traits, skills	BJC assignments created to reflect computational thinking skills needed in future

grading and limit the effects of irrelevant variance. As CS Principles is marketed as a creative course, the rubrics are designed to be fair and allow variance in content of student submissions while still measuring quality.

*Cognitive Complexity* The cognitive complexity criterion helps us address a concern we had with using task-based rubrics to grade novice student programs. Task-based rubrics measure whether the submitted program exactly meets requirements, and rewarded efficiency, but did not allow for a fine-grained analysis of the learning objectives the students may have demonstrated successfully in their programs. The initial task-based rubrics provided a good base for teachers to check student work, however, differences in implementation and algorithmic choice were not registered with the task-based rubric. The new rubrics grew from an expansion of the logic category of the initial rubric format, so that teachers could more easily identify evidence of learning goals in student assignments. The new learning-based rubric categories often focus on algorithmic design, logic flow and other aspects of computational thinking. By basing our assessment measures on more abstract concepts as opposed to final product output, we are ensuring that our assessment can better reflect the cognitive and algorithmic decisions made by students without losing artificial points, thus capturing a greater level of cognitive complexity than the previous rubrics.

*Meaningfulness* This is another category where we focus on the teacher perspective over the student one. In the initial implementation of CS Principles as an NCSU college course (CSC 200), student lab assignments were graded simply as pass or fail. There was no metric available to the numerous TAs about how well a student completed the task. As very few of the TAs had a computing

background, projects that were later rated low-quality by our system, were initially given scores of 100. This lack of valuable finer-grained grading feedback limited both students and professors in understanding how students were progressing in the course. The learning-based rubrics can help teachers have more confidence in whether or not their lessons are promoting student learning. Increased confidence in teaching the course correlates with higher retention rates for those teachers, and consequently increases adoption rates by their community peers as the new course seems less intimidating [Ni09].

*Costs & efficiencies* Using traditional rubrics as a support tool for beginning CS Principles teachers is very cost effective. Unlike electronic systems and new software platforms, teachers are very comfortable with rubrics. School administration is also on board with teachers using rubrics. Our rubrics address efficiencies by having support from key players in the K-12 school system, allowing us to more easily and quickly provide needed support to novice teachers. We also want rubrics to be created by knowledgeable persons who are familiar with both computing and the CS Principles course. Gathering together the Master teachers is costly in terms of their limited time commitment and transportation funding needs. Compared to the geo-separated Delphi method for rubric development, NGT is a much quicker process [Cat18]. Similarly, our past studies have shown that well-trained computing undergrads are just as reliable as our active BJC teachers [Cat16], thus our rubric creation process is more cost effective than a geo-separated Delphi or an expensive in-person meeting of Master teachers. Although this process is not as efficient as a single well-trained teacher creating personal rubrics for their own classroom, it does help address the widespread lack of computing experience in our beginning teachers. As teachers become more comfortable with the content, they will have a foundation on which to base their future materials.

*Educational Consequences* Although this facet of quality has not been fully explored in relation to our study, based on similar research, we can expect to see positive educational outcomes. According to work by Ericson, successful teachers use assessment to determine student misconceptions, give explanatory feedback, and have students use grading rubrics to understand how open-ended problems are graded [Eri14]. Students who have the scoring metrics made available to them beforehand are provided clear goals for their work and are more inclined to learn the assessed materials [Art00]. Additionally, as these rubrics are formative they help faculty recognize where students are struggling and to address problems immediately, as compared to the final AP exams. These rubrics provide a base of understanding for intended learning goals of the unit so that teachers can better reflect on their teachings and adapt as needed to focus on the essential knowledge [JS07]

**Table 8.4** Evidence for Wheel of Competency Assessment pt. III: Term, Definition, Evidence

Meaningfulness	Fitness for Self-Assessment	Costs & Efficiencies	Educational Consequences
The assessment should have a significant value for both teachers and learners	Assessment assists in self-regulated learning; makes the criterion clear, shows weaknesses and stimulates reflection on the learning process	Time and resources needed are justified by positive effects: improvements in learning & teaching	How does the assessment cause teachers & learners to adjust their learning activities?
Meaningful feedback; More confidence assessing student learning, teaching [Ni09]	Not Tested - Out of Scope	Study 5, modified NGT method greatly reduces prior costs & time requirements	Determine student misconceptions; student preparation; increased inclination to practice desired goals; teaching aide [Eri14; Art00; JS07]

## 8.5 Discussion

In this study, we set out to validate the CS Principles/BJC learning-based rubrics using principles from the education community. After investigating validity as three separate categories [Mes80], and then again as 6 aspects of a single construct validity [Mes96; Ass99], we decided that the best route for assessing the quality of our rubrics was actually the Wheel of Competency Assessment [Baa06]. Unlike the prior two strategies that were designed for large-scale standardized testing, the Wheel of Competency Assessment framework was developed for complex competency assessment programs that take on various forms of implementation. The quality criteria presented in the framework still measure aspects of the prior validity definitions, but also take on new measurements for practical use in lower-stakes classroom evaluations.

When comparing the suite of scoring rubrics and their development against the Wheel of Competency Assessment, our findings provide support to satisfactorily meeting 10 out of the 12 criterion including the 5 core criteria that Baartman suggests must be met before considering criteria in the outer layers [Baa06]. There are two criteria that we have not fully tested. These criteria are the fitness for student self-assessment and the student aspect of educational consequences. Otherwise, we demonstrate consistency, fairness, authenticity, and each of the other quality criteria supporting the validity of these rubrics. The lack of student reporting is due to our targeted examination of

teachers who are new to computing. There are a large number of K-12 teachers being mobilized to teach either a new computing course or to add computing into their existing course. These teachers do not always have a full understanding of the materials they are teaching and what the outcomes should be. Our main purpose was to provide support to the curriculum such that it could be picked up by a beginner teacher and used in the classroom more confidently.

After releasing our rubrics we have found the number of teacher requests for assessments have declined. Our Summer 2017 study demonstrated that teachers were able to program and grade assignments with the rubrics as support. We anecdotally observed that providing the rubrics helped teachers feel more confident in their ability to perform the necessary duties in the classroom. Ni's 2009 study on factors influencing adoption, shows that when teachers are more confident in their ability to teach the curriculum, they are more likely to support the spread and adoption of the course [Ni09], which is critical to the systematic acceptance of computing curricula in the K-12 classroom.

There are limitations to this work, as some evidence relies on the opinions of a few key persons. Both our original Delphi groups and our hybrid NGT team were comprised of selected individuals. These groups had a large effect on the outcome of the rubrics developed. Furthermore, although we performed reliability testing with three separate groups, this was still a small sample size. We believe that due to the low-stakes nature of the rubrics, the sampling sizes will not impair the overall goal and outcomes of this endeavor.

Other limitations include the use of self-reported feedback from the teachers in professional development. Research has shown that participants want to respond in a way that appeases researchers by making themselves look as good as possible. Thus, they tend to under-report behaviors deemed inappropriate by researchers or other observers, and they tend to over-report behaviors viewed as appropriate [DGV02]. There could be several reasons for this in our study, including the perceived opportunity for more funding, the desire for more resources, or other outside influences.

## **8.6 Conclusions**

The lack of qualified computer scientists available to fill positions in the tech industry is parallel to a lack of qualified computer science teachers at the K-12 level. With both parents and administration pushing the need to teach students the necessary skills for 21st century technology jobs [Goo16], these teachers need both professional development as well as complete curriculum from which to pick and choose modules in order to gain confidence in teaching computer science content and to establish a sense of ownership of the course. As handing over lessons without meaningful ways to assess student learning can lead to teacher confusion, finding a systematic way to quickly generate a full suite of rubrics for new computing courses is imperative to the adoption of computational

thinking in K-12 classrooms.

To evaluate our learning-based rubrics designed for a blocks-based Computer Science Principles course, we aligned our process to the Wheel of Competency Assessment. Using the wheel as a measurement of quality, we found that our well-defined and task-specific rubrics were able to meet 10 out of twelve criteria, and met the teacher perspective on an 11th of the criteria for quality problem based assessments. This combination of findings provides support that the developed rubrics demonstrate key aspects of reliability, validity, and other important concepts such as meaningfulness and acceptability.

The standard validity and reliability metrics used for classical high stakes testing are not appropriate for use for formative feedback and assessment in new computing classes taught by computing novices. Instead, Baartman's Wheel of of Competency Assessment should be used for low-stakes assessments especially rubrics; because of ease of use and understanding by non-education researchers or statisticians. Baartman's wheel also acts as a tool for checking alignment between the intentions of the assignment and how it is assessed. We have therefore applied the Wheel of Competency Assessment model to obtain measures of acceptance and practicality for our rubrics. The rubrics were developed using a modified Nominal Group Technique with trained undergrads as a cost-effective alternative to the having well-versed computing teachers leading new K-12 computing courses. This process is not as ideal as having an abundance of already trained K-12 computing teachers available to create brand new materials and assessments. However, the Wheel of Competency demonstrates that the created rubrics are suitable for low-stakes assessments in AP CS Principles. This is particularly important, because not only are the teachers new to computing, but there are very few rubrics made for novice programming that are aligned to learning objectives.

Taken together, these findings suggest a role for our process in promoting the rapid production of reliable and valid rubrics as formative support for beginner computing teachers in the frequently changing landscape of K-12 computing education while maintaining a high level of cost-effectiveness with measurably good outcomes.

## **8.7 Future Work**

As the current system hasn't been vetted for student fitness of self-assessment or educational consequence from the student's perspective, these would be useful avenues to round out the evaluations using the Wheel of Competency Analysis. Our hypotheses are that (1) the rubrics allow students to self-assess through understandable language and meaning, and (2) that students using the rubrics as a guideline would have a higher presence of the requested learning outcomes expressed in code artifacts demonstrating positive educational consequences.

In order to assess the fitness for self-assessment and the educational consequences for student use of the rubrics it is advised to run a study with students from active CS Principles classrooms. In this study, one classroom set of students would be given the rubric at the onset of the lab activity to use as a guide for completion, the second group would be given the rubric after the lab assignment. Both groups would be tasked to give themselves an expected grade based-on the rubric before turning in the assignment to the instructor. In order to minimize demand characteristics, or 'the good-subject' effect where participants give the answers they think researchers are looking for [NM08], students will be given 2 points of extra credit for estimating their grade correctly (within 1 point of final grade). Furthermore, a sample student interviews would be conducted throughout the programming activity. Investigators would focus on questions pertaining to what students think the desired outcomes are and how they would go about achieving those. Furthermore, follow-up interviews would be conducted with samples of students who both estimated their grades correctly and with those who were not able to estimate their grade well.

In order to answer hypothesis 1, we would examine students' expected vs. actual grade, as well as statements made in student interviews. The distribution of students' expected grades to actual grades would provide empirical evidence to students' ability to self-assess. The interview statements would provide richer context to the analysis, especially for those not meeting our hypothesis. Student insights on why they might have misjudged their projects could allow us to make necessary revisions to the final rubrics. Furthermore, to answer hypothesis 2, we would examine differences between groups of how well students completed the task. We would expect that students using the rubrics as a guideline would have a higher presence of the requested learning outcomes expressed in code artifacts than those who were not given access to the rubrics beforehand. The differences in frequency of expected learning outcomes would provide evidence to the educational consequences aspect of WoCA. To further investigate this aspect, we need to identify the unexpected consequences of rubric use. This can be done through the structured open-ended questions in the student follow-up interviews as well as through classroom observations while students are completing the labs. Through completion of this final study, a complete analysis of quality using the Wheel of Competency Assessment can be performed.

## CHAPTER

# 9

# CONCLUSIONS

## 9.1 Review

In this chapter, I summarize how the work in the previous chapters contributed to validating my hypotheses and answering the research questions outlined in the introduction. To recap, my research questions and hypotheses are as follows:

## 9.2 Research Questions

- RQ1 How can we help teachers better understand learning objectives for labs and identify whether they are achieved in student artifacts?
- RQ2 How can we modify educational psychology methods to meet our needs for creating a large set of rubrics for classroom settings?
- RQ3 How well do teachers identify computational thinking in student artifacts using task-based and learning-based rubrics?

## 9.3 Hypotheses

- H1 We can use the Delphi method to create learning-based rubrics that perform better than traditional task-based rubrics at helping teachers grade assignments meaningfully.
- H2 Use of the Delphi method to create rubrics will lead to sufficient levels of inter-rater reliability among novice graders on low-stakes assessments for programming lab assignments.
- H3 We can use a modified NGT approach to efficiently and quickly produce a full suite of quality assured rubrics for BJC lab assignments.
- H4 Learning-based rubrics will help support beginning CS Principles teachers consistently assess important computational thinking elements in student code.
- H5 We can apply the Wheel of Competency Assessment to show that the created rubrics are appropriate for low-stakes assessment of CS Principles labs.

### Research Question 1

Research question 1 is “How can we help teachers better understand learning objectives for CS Principles labs and identify whether these learning objectives are achieved in student artifacts?” In my early investigations, I learned that many teachers did not feel equipped to create their own rubrics, limiting the quality of feedback they could give to their students. In my pilot study, I created rubrics based on grading trends in auto-graders for introductory computing courses [Cat16]. I tested these rubrics by having both an experienced computer science grader and a new CS Principles grader use the rubrics. I learned that although both graders were able to grade consistently with each other, there was still room for improvement to expand the learning focus of the rubrics. Research into rubric design strategies encouraged a redesign of the rubrics from being task oriented to being learning oriented. However, as the CS Principles labs we were working with did not have learning objectives explicitly assigned to them, we would need to find a way to identify appropriate learning objectives. Using the newly-created learning oriented rubrics, teachers were better able to identify evidence of the learning objectives in student assignments as seen in Chapter 4.

### Research Question 2

Research question 2 is “How can we modify education psychology methods to meet our needs for creating a large set of rubrics for classroom settings?” After implementing both a full scale

Delphi study and a modified local Delphi study (Chapter 5), our results followed those of previous researchers. The panelists were able to systemically agree upon learning objectives for their particular labs. However, the national Delphi with selected experts was both time and cost prohibitive. The local Delphi on the other hand, did not accrue direct cost on our end and finished in a very timely manner. Unfortunately, to replicate the study was not as feasible for each rubric set, as getting the master teachers together during the school year is difficult. In order to effectively address these issues, we switched to a Nominal Group Technique approach and utilized trained undergraduate computer science students as the panel experts (Chapter 6). This both formalized the in-person procedures and gave us more effective access to qualified panelists. Using our final NGT method allowed us to create a full set of task-based and learning-based rubrics within a single semester. When assessing these rubrics for reliability and quality we were also able to establish consistency between active CS Principles teachers (Chapter 7) and ensure a level of quality using the Wheel of Competency Assessment (Chapter 8).

### **Research Question 3**

Research question 3 is “How well do teachers identify computational thinking in student artifacts using task-based and learning-based rubrics?” In our pilot study, the experienced computer science grader and the novice grader were able to achieve a satisfactory level of inter-rater reliability only after two rounds of training and revisions to the ambiguity of the initial task-based rubric. When comparing the task-based rubric to the learning-based rubric however, our novice rater was more confident in grading with the learning-based rubric and being able to consistently grade student projects, particularly borderline cases that deviate from the typical solution.

Furthermore, when testing with active CS Principles teachers with <2 years teaching experience in the course, we were able to identify which aspects of computational thinking they were able to recognize. Our study in Chapter 7 demonstrated that teachers could recognize concepts such as abstraction, parameters, mathematical operations, and styling, however ratings in categories like logic statements and appropriate naming conventions were less consistent. In Chapter 7 we make recommendations on how to address these issues.

### **Hypothesis 1**

Hypothesis H1 is that “We can use the Delphi method to create learning-based rubrics that perform better than traditional task-based rubrics at helping teachers grade assignments meaningfully.” H1 is supported by work in Chapters 3 and 4. This work shows that raters using both task-based and

learning-based rubrics were able to achieve a sufficient level of inter-rater reliability. However, raters using the task-based rubrics required multiple revisions to the initial rubrics, which did not occur when raters used the learning-based rubrics. Furthermore, we found that the task-based rubrics, mirroring traditional CS grading schemes, bias high performing students, giving more credit to more elegant and efficient solutions, which are not appropriate measures of success for novices. The learning-based rubrics on the other hand, are grounded in the learning outcomes expected from each lab, and this means that by simply achieving the list of lab requirements, a student has demonstrated the intended level of competency for the AP CS Principles course. Therefore we suggest that learning-based rubrics are better than the task-oriented ones at helping teachers provide meaningful feedback to beginner computer science students.

## **Hypothesis 2**

Hypothesis H2 is that "Use of the Delphi method to create rubrics will lead to sufficient levels of inter-rater reliability among novice graders on low-stakes assessments for programming lab assignments." H2 is supported by work in Chapter 5 where we had both Master teachers and undergraduate computing novices grade 60+ student code samples using the learning based rubrics. From this study we found novice raters to achieve a inter-rater reliability measure of .78 to .83 which matches the Master teacher reliability of .79. Because the novice graders align with Master score and because each of the reliability estimates are above the commonly accepted satisfactory threshold of .70 we find this hypothesis to be true.

## **Hypothesis 3**

Hypothesis H3 is that "We can use a modified NGT approach to efficiently and quickly produce a full suite of quality assured rubrics for BJC lab assignments." Although the Delphi method produced rigorous quality results, the costs were too high to use this method to mass-produce rubrics for every BJC lab assignment. We support H3 in Chapters 6 and 7 where we streamline the rubric creation process using a team of 'almost experts' to generate 66 rubrics (32 learning-based and 34 task-based) within the course of ten 60-90 minute sessions. This greatly surpasses the 11-week time span to carry out our original more traditional Delphi method which only produced a single rubric. To assess the quality of these rubrics we tested them with active CS Principles teachers and found them to have the same level of consistency in inter-rater reliability, ICC = .75 for both teachers and STEM education students, and ICC of up to .89 for just teachers on moderate level assignments. Therefore, we find our modified NGT to be a suitable replacement for the previously used Delphi.

## **Hypothesis 4**

Hypothesis H4 is that “Learning-based rubrics will help support beginning CS Principles teachers consistently assess important computational thinking elements in student code.” H4 is supported by work in Chapter 7 where we assessed the reliability of the new rubrics by having active CS Principles teachers identify student code areas that influenced their grading decisions for each category of the rubrics. We found that teachers were able to identify computational evidence of abstraction and stylizing which are less defined terms, in addition to the expected parameters and mathematical operations. We also concluded that there is still room for improvement – for example, participants did not consistently identify regions for appropriate naming. This concept requires teachers to understand the intentions of the variables and custom blocks to determine whether they are well named. As code deviates from a typical solution, by either being more creative or less detailed, the teachers were less apt to understand the underlying programming logic. We believe that although the rubrics help significantly in teacher grading, additional resources such as annotated solution files might also be beneficial.

## **Hypothesis 5**

Hypothesis H5 is that “We can apply the Wheel of Competency Assessment to show that the created rubrics are appropriate for low-stakes assessment of CS Principles labs.” H5 is supported by work in Chapter 8 where we align our rubrics and creation process to the Wheel of Competency Assessment directly. This work shows that we were able to provide sufficient evidence for 10 out of 12 of the quality criteria and met the teacher perspective on an 11th of the criteria for quality problem based assessments. The remaining 1.5 aspects that we did not test were the fitness for student self-assessment and the student perspective for educational consequences. These were left out as the primary focus of this research is on teacher support; according to our comparison using Baartman’s wheel, we have accomplished.

## **9.4 Contributions**

In conclusion, this research presents the following contributions:

1. I conducted the first application of the Delphi Method to create content-validated rubrics for a K-12 computing course. Using these methods we were able to gain insights on streamlining a robust process for creating a large quantity of quality rubrics, which benefits the

rapidly expanding CS Principles program and can be applied to other newly developed novice computing courses.

2. I redesigned the rubric creation process to be more cost-effective and thus feasible to replicate for use in real-life applications and school settings. This allows for quick distribution of rubrics on rapidly changing courses. As computing trickles down the K-12 pipeline, we will be presented with more teachers from diverse backgrounds and course content that will have to adjust as students in lower grades become more experienced. Thus, being able to generate new reliable rubrics quickly and so that they support novice teachers will be beneficial.
3. I proved that the rubrics created with the new modified NGT process and almost-experts are as reliable as the initial Delphi generated rubrics. As our modified NGT method was more cost-effective and efficient due to using surrogate experts, it was critical to ensure that they held up to the same standards as the initial Delphi rubrics. Ensuring a level of quality assessment to our rubrics is important for their adoption by teachers.
4. I devised a novel application of the Baartman's Wheel of Competency Assessment to establish a high level of validity, value and acceptability of the rubrics. By aligning our rubrics to the Wheel of Assessment we were able evaluate whether these rubrics met their intended goals. As we used new tools and open-ended processes to assess student work, it is important to evaluate their appropriateness for use by new CS Principles teachers.
5. I led a junior research team composed of upper classification computer science undergraduates in generating 32 Beauty & Joy of Computing rubrics designed to measure implementation of learning objectives in student lab assignments. This is the first such system, as well as the first such system specifically designed for beginning teachers.

## **9.5 Future Work**

In terms of directions for future research in helping teachers manage efficient and effective grading for their courses, further work could involve a deeper analysis of automating computation thinking identification in code so that assignments can be quickly graded by AI and then reviewed by a teacher for accuracy. Another possible area of future development that goes along with this would be to develop or integrate with a dashboard system that allows teachers to quickly grade Snap programming environments using an online environment. Some of our BJC teachers are teaching course loads with 150 students, and although this is common in universities with plenty of teaching assistants available, at the K-12 level there just are not enough resources. We found Gradescope

to be very useful for our tagging and analysis, however the setup process and final metrics were still crude. With a more streamlined process for code analysis, running studies on both student and teacher identification of computational thinking will prove easier. Furthermore, the current system hasn't been vetted for student fitness of self-assessment or educational consequence from the student's perspective. These would be useful avenues to round out the evaluations using the Wheel of Competency Analysis as outlined in Chapter 8.

## BIBLIOGRAPHY

- [AS96] Abelson, H. & Sussman, G. J. *Structure and Interpretation of Computer Programs*. 2nd. Cambridge, MA, USA: MIT Press, 1996.
- [ACM15] ACM. “Special Section: The Role of Programming in a Non-Major, CS Course”. *ACM Inroads* **6.1** (2015), pp. 42–62.
- [AW11] Adams, W. K. & Wieman, C. E. “Development and validation of instruments to measure learning of expert-like thinking”. *International Journal of Science Education* **33.9** (2011), pp. 1289–1312.
- [AZ96] Adler, M. & Ziglio, E. *Gazing into the oracle: The Delphi method and its application to social policy and public health*. Jessica Kingsley Publishers, 1996.
- [Aho12] Aho, A. V. “Computation and computational thinking”. *The Computer Journal* **55.7** (2012), pp. 832–835.
- [AK09] Ahoniemi, T. & Karavirta, V. “Analyzing the Use of a Rubric-based Grading Tool”. *Proceedings of the 14th Annual ACM SIGCSE Conference on Innovation and Technology in Computer Science Education*. ITiCSE ’09. New York, NY, USA: ACM, 2009, pp. 333–337.
- [Art00] Arter, J. “Rubrics, Scoring Guides, and Performance Criteria: Classroom Tools for Assessing and Improving Student Learning.” (2000).
- [Ass99] Association, A. E. R. et al. *Standards for educational and psychological testing*. American Educational Research Association, 1999.
- [Baa06] Baartman, L. K. et al. “The wheel of competency assessment: Presenting quality criteria for competency assessment programs”. *Studies in Educational Evaluation* **32.2** (2006), pp. 153–170.
- [Bak96] Baker, E. L. et al. “Dimensionality and generalizability of domain-independent performance assessments”. *The Journal of Educational Research* **89.4** (1996), pp. 197–205.
- [Bar84] Bardecki, M. J. “Participants’ response to the Delphi method: An attitudinal perspective”. *Technological Forecasting and social change* **25.3** (1984), pp. 281–292.
- [Bar16] Barnes, T. et al. “Scaling Up for CS10K: Teaching and Supporting New Computer Science High School Teachers”. *Proc. 47th ACM Tech. Symp. on CS Ed. SIGCSE ’16*. Memphis, Tennessee, USA: ACM, 2016, pp. 720–720.
- [BS11] Barr, V. & Stephenson, C. “Bringing Computational Thinking to K-12”. *ACM Inroads* (2011), pp. 48–54.

- [Bec03] Becker, K. “Grading programming assignments using rubrics”. *ACM SIGCSE Bulletin*. Vol. 35. 3. ACM. 2003, pp. 253–253.
- [Ben15] Bender, E. et al. “Identifying and formulating teachers’ beliefs and motivational orientations for computer science teacher education”. *Studies in Higher Education* (2015), pp. 1–16.
- [BD16] Blaheta, D. & Decker, A. “Rubricing Like a Boss: Writing and Using Rubrics For Faster, Fairer Grading of Student Assignments (Abstract Only)”. *Proceedings of the 47th ACM Technical Symposium on Computing Science Education*. SIGCSE ’16. New York, NY, USA: ACM, 2016, pp. 715–716.
- [Bos02] Boston, C. *Understanding Scoring Rubrics: A Guide for Teachers*. ERIC, 2002.
- [BR12] Brennan, K. & Resnick, M. “New frameworks for studying and assessing the development of computational thinking”. *Proceedings of the 2012 annual meeting of the American Educational Research Association, Vancouver, Canada*. 2012.
- [Bro13] Brookhart, S. M. *How to create and use rubrics for formative assessment and grading*. ASCD, 2013.
- [Bro79] Brooks, K. W. “Delphi technique: Expanding applications.” *North Central Association Quarterly* **53.3** (1979), pp. 377–85.
- [Bus01] Bush, G. W. “No Child Left Behind.” (2001).
- [Cab01] Cabaniss, K. “Counseling and Computer Technology in the New Millennium—An Internet Delphi Study”. PhD thesis. 2001.
- [CF59] Campbell, D. T. & Fiske, D. W. “Convergent and discriminant validation by the multitrait-multimethod matrix.” *Psychological bulletin* **56.2** (1959), p. 81.
- [Can15] Canterbury, N. University of. *CS Unplugged: Computer Science without a Computer*. web. url=csunplugged.org. 2015.
- [Cat18] Cateté, V. “A Streamlined Approach to the Systematic Creation of Rubrics for Computer Science Principles”. *Proceedings of the 2018 ACM SIGCSE Technical Symposium on Computer Science Education*. SIGCSE ’18. New York, NY, USA: ACM, 2018.
- [CB17] Cateté, V. & Barnes, T. “Application of the Delphi Method in Computer Science Principles Rubric Creation”. *Proceedings of the 2017 ACM Conference on Innovation and Technology in Computer Science Education*. ITiCSE ’17. New York, NY, USA: ACM, 2017, pp. 164–169.

- [Cat16] Cateté, V. et al. “Developing a Rubric for a Creative CS Principles Lab”. *Proceedings of the 2016 ACM Conference on Innovation and Technology in Computer Science Education. ITiCSE '16*. New York, NY, USA: ACM, 2016, pp. 290–295.
- [Cec09] Ceci, S. J. et al. “Women’s underrepresentation in science: sociocultural and biological considerations.” *Psychological bulletin* **135.2** (2009), p. 218.
- [CC04] Cheng, L. & Curtis, A. “Washback or backwash: A review of the impact of testing on teaching and learning”. *Washback in language testing: Research contexts and methods* (2004), pp. 3–17.
- [Cic94] Cicchetti, D. V. “Guidelines, criteria, and rules of thumb for evaluating normed and standardized assessment instruments in psychology.” *Psychological assessment* **6.4** (1994), p. 284.
- [Cla11] Clayton, S. *Microsoft Research: How fighting email spam is helping the search for an HIV vaccine @ONLINE*. 2011.
- [Col97] College Board. *AP Data - Archived Data*. web. 1997-2013.
- [Col14] College Board. “AP Computer Science Principles Curriculum Framework”. *AP Program* (2014).
- [Col15] College Board. *Advances in AP: AP Computer Science Principles*. web. 2015.
- [Col17] College Board. *Additional Curricula and Pedagogical Support - Advances in AP - The College Board*. 2017. URL: <https://advancesinap.collegeboard.org/stem/computer-science-principles/curricula-pedagogical-support>.
- [Cod] *Computer Science Principles | Code.org*. 2017. URL: <https://code.org/educate/csp>.
- [CG65] Cronbach, L. J. & Gleser, G. C. “Psychological tests and personnel decisions.” (1965).
- [CM01] Crouch, C. H. & Mazur, E. “Peer instruction: Ten years of experience and results”. *American Journal of Physics* **69.9** (2001), pp. 970–977.
- [CST08] CSTA Tacher Certification Task Force. *Ensuring Exemplary Teaching in an Essential Discipline: Addressing the Crisis in Computer Science Teacher Certification*. Tech. rep. New York: Computer Science Teachers Association, 2008.
- [Cun14] Cuny, J. et al. “CS Principles Professional Development: Only 9,500 to Go!” *Proceedings of the 45th ACM Technical Symposium on Computer Science Education. SIGCSE '14*. New York, NY, USA: ACM, 2014, pp. 543–544.

- [DH63] Dalkey, N. & Helmer, O. “An Experimental Application of the Delphi Method to the Use of Experts”. English. *Management Science* **9.3** (1963). see abstract, pp. 458–467.
- [Dan12] Danielsiek, H. et al. “Detecting and Understanding Students’ Misconceptions Related to Algorithms and Data Structures”. *Proceedings of the 43rd ACM Technical Symposium on Computer Science Education*. SIGCSE ’12. New York, NY, USA: ACM, 2012, pp. 21–26.
- [Del86] Delbecq, A. et al. *Group techniques for program planning: a guide to nominal group and Delphi processes*. Middleton, WI: Green Briar Press, 1986.
- [DGV02] Donaldson, S. I. & Grant-Vallone, E. J. “Understanding Self-Report Bias in Organizational Behavior Research”. *Journal of Business and Psychology* **17.2** (2002), pp. 245–260.
- [Emb83] Embretson, S. “Construct Validity: Construct Representation Versus Nomothetic Span”. *Psychological Bulletin* **93.1** (1983), pp. 179–197.
- [Eng15] Engineering Online. *CSC 216 Programming Concepts - Java*. web. 2015.
- [Eri14] Ericson, B. J. et al. “Preparing Secondary Computer Science Teachers Through an Iterative Development Process”. *Proceedings of the 9th Workshop in Primary and Secondary Computing Education*. WiPSCE ’14. New York, NY, USA: ACM, 2014, pp. 116–119.
- [Eug16] Eugene, K. et al. “The Usefulness of Rubrics in Computer Science”. *J. Comput. Sci. Coll.* **31.4** (2016), pp. 5–20.
- [Fit13] Fitzgerald, S. et al. “What are we thinking when we grade programs?” *Proceeding of the 44th ACM technical symposium on Computer science education*. ACM. 2013, pp. 471–476.
- [Fra13] Franklin, D. et al. “Assessment of Computer Science Learning in a Scratch-based Outreach Program”. *Proceeding of the 44th ACM Technical Symposium on Computer Science Education*. SIGCSE ’13. New York, NY, USA: ACM, 2013, pp. 371–376.
- [Fri01] Friend, J. G. “A Delphi study to identify the essential tasks and functions for ADA coordinators in public higher education”. PhD thesis. University of Missouri-Columbia, 2001.
- [GES10] Gal-Ezer, J. & Stephenson, C. “Computer Science Teacher Preparation is Critical”. *ACM Inroads* **1.1** (2010), pp. 61–66.
- [Gar11] Garcia, D. et al. “FRABJOUS CS – Framing a Rigorous Approach to Beauty and Joy for Outreach to Underrepresented Students in Computing at Scale”. 2011.

- [Gla78] Glaser, B. G. *Theoretical sensitivity: Advances in the methodology of grounded theory*. Sociology Pr, 1978.
- [Gol08] Goldman, K. et al. "Identifying Important and Difficult Concepts in Introductory Computing Courses Using a Delphi Process". *Proceedings of the 39th SIGCSE Technical Symposium on Computer Science Education*. SIGCSE '08. New York, NY, USA: ACM, 2008, pp. 256–260.
- [Gol10] Goldman, K. et al. "Setting the scope of concept inventories for introductory computing subjects". *ACM Transactions on Computing Education (TOCE)* **10.2** (2010), p. 5.
- [Gom85] Gomez, C. C. "Emerging Curricula for Computer Science". English. Copyright - Copyright UMI - Dissertations Publishing 1985; Last updated - 2014-01-10; First page - n/a. PhD thesis. Arizona State University, 1985, 217–217 p.
- [Goo14] Goode, J. et al. "Curriculum is not enough: the educational theory and research foundation of the exploring computer science professional development model". *Proceedings of the 45th ACM technical symposium on Computer science education*. ACM. 2014, pp. 493–498.
- [Goo16] Google Inc. & Gallup Inc. *Trends in the State of Computer Science in U.S. K-12 Schools*. 2016.
- [Gra14] Gradescope. *Save time grading*. 2014.
- [GM09] Groth-Marnat, G. *Handbook of psychological assessment*. John Wiley & Sons, 2009.
- [Gul04] Gulikers, J. T. et al. "A five-dimensional framework for authentic assessment". *Educational technology research and development* **52.3** (2004), pp. 67–86.
- [Gus73] Gustafson, D. H. et al. "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), pp. 280–291.
- [Hak98] Hake, R. R. "Interactive-engagement versus traditional methods: A six-thousand-student survey of mechanics test data for introductory physics courses". *American journal of Physics* **66.1** (1998), pp. 64–74.
- [Ham96] Hambleton, R. K. "Advances in assessment models, methods, and practices". *Handbook of educational psychology* **899925** (1996).
- [Har14] Harvey, B. et al. "Snap! (Build Your Own Blocks) (Abstract Only)". *Proceedings of the 45th ACM Technical Symposium on Computer Science Education*. SIGCSE '14. Atlanta, Georgia, USA: ACM, 2014, pp. 749–749.

- [HH12] Harvey, N. & Holmes, C. A. “Nominal group technique: An effective method for obtaining group consensus”. *International Journal of Nursing Practice* **18.2** (2012), pp. 188–194.
- [HH10] Herman, G. L. & Handzik, J. “A preliminary pedagogical comparison study using the digital logic concept inventory”. *Frontiers in Education Conference (FIE), 2010 IEEE*. IEEE. 2010, F1G–1.
- [Hes92] Hestenes, D. et al. “Force concept inventory”. *The physics teacher* **30.3** (1992), pp. 141–158.
- [Hic13] Hicks, A. “BOTS: Harnessing player data and player effort to create and evaluate levels in a serious game”. *Educational Data Mining 2013*. 2013.
- [JU97] Jackson, D. & Usher, M. “Grading Student Programs Using ASSYST”. *SIGCSE Bull.* **29.1** (1997), pp. 335–339.
- [JS07] Jonsson, A. & Svingby, G. “The use of scoring rubrics: Reliability, validity and educational consequences”. *Educational Research Review* **2.2** (2007), pp. 130–144.
- [Jud72] Judd, R. C. “Use of Delphi methods in higher education”. *Technological Forecasting and Social Change* **4.2** (1972), pp. 173–186.
- [KW14] Karpierz, K. & Wolfman, S. A. “Misconceptions and concept inventory questions for binary search trees and hash tables”. *Proceedings of the 45th ACM technical symposium on Computer science education*. ACM. 2014, pp. 109–114.
- [Klo14] Kloser, M. “Identifying a core set of science teaching practices: A delphi expert panel approach”. *Journal of Research in Science Teaching* **51.9** (2014), pp. 1185–1217.
- [Koh10] Koh, K. H. et al. “Towards the automatic recognition of computational thinking for adaptive visual language learning”. *Visual Languages and Human-Centric Computing (VL/HCC), 2010 IEEE Symposium on*. IEEE. 2010, pp. 59–66.
- [Kot13] Kothiyal, A. et al. “Effect of Think-pair-share in a Large CS1 Class: 83% Sustained Engagement”. *Proceedings of the Ninth Annual International ACM Conference on International Computing Education Research*. ICER '13. New York, NY, USA: ACM, 2013, pp. 137–144.
- [Lec84] Lecklitner, G. L. “Protecting the rights of mental patients: A view of the future”. PhD thesis. Ohio State University, 1984.
- [Len56] Lennon, R. T. “Assumptions underlying the use of content validity”. *Educational and Psychological Measurement* **16.3** (1956), pp. 294–304.

- [LT75] Linstone, H. & Turoff, M. *The Delphi method: techniques and applications*. Advanced book program: Addison-Wesley. Reading, MA: Addison-Wesley Pub. Co., Advanced Book Program, 1975.
- [Loe57] Loevinger, J. "Objective tests as instruments of psychological theory". *Psychological reports* **3.3** (1957), pp. 635–694.
- [Mac11] Mack, N. C.-G. *A Research Study Using the Delphi Method to Define Essential Competencies for a High School Game Art and Design Course Framework at the National Level*. ERIC, 2011.
- [Mar08] Margolis, J. *Stuck in the Shallow End: Education, Race, and Computing*. The MIT Press, 2008.
- [Mar91] Martorella, P. H. "Consensus building among social educators: A Delphi study". *Theory & Research in Social Education* **19.1** (1991), pp. 83–94.
- [McC03] McCauley, R. "Rubrics as assessment guides". *ACM SIGCSE Bulletin* **35.4** (2003), pp. 17–18.
- [McM16] McMillan, S. S. et al. "How to use the nominal group and Delphi techniques". *International Journal of Clinical Pharmacy* **38.3** (2016), pp. 655–662.
- [Mes96] Messick, S. "Technical Issues in Large-Scale Performance Assessment". ERIC, 1996. Chap. Validity of Performance Assessments.
- [Mes80] Messick, S. "Test Validity and the Ethics of Assessment." *American Psychologist* **35.11** (1980), pp. 1012–27.
- [Mes89] Messick, S. "Validity." (1989).
- [Mod12] Modi, K. et al. "Generation STEM: What Girls Say about Science, Technology, Engineering and Math". *A Report from the Girl Scout Research Institute*. New York, NY: Girl Scouts of the USA (2012).
- [Mor13] Morelli, R. et al. "Teaching the CS Principles Curriculum with App Inventor (Abstract Only)". *Proceeding of the 44th ACM Technical Symposium on Computer Science Education*. SIGCSE '13. Denver, Colorado, USA: ACM, 2013, pp. 762–762.
- [Mor15] Morelli, R. et al. "Analyzing Year One of a CS Principles PD Project". *Proceedings of the 46th ACM Technical Symposium on Computer Science Education*. SIGCSE '15. Kansas City, Missouri, USA: ACM, 2015, pp. 368–373.

- [Mos03] Moskal, B. M. “Developing Classroom Performance Assessments and Scoring Rubrics-Part I & II. ERIC Digest.” (2003).
- [ML00] Moskal, B. M. & Leydens, J. A. “Scoring rubric development: Validity and reliability”. *Practical assessment, research & evaluation* **7.10** (2000), pp. 71–81.
- [MJH95a] Murry Jr, J. W. & Hammons, J. O. “Assessing the managerial and leadership ability of community college administrative personnel”. *Community College Journal of Research and Practice* **19.3** (1995), pp. 207–216.
- [MJH95b] Murry Jr, J. W. & Hammons, J. O. “Delphi: A versatile methodology for conducting qualitative research”. *The Review of Higher Education* **18.4** (1995), pp. 423–436.
- [Nat10] National Research Council. “Committee for the Workshops on Computational Thinking: Report of a Workshop on The Scope and Nature of Computational Thinking” (2010).
- [Ni09] Ni, L. “What Makes CS Teachers Change?: Factors Influencing CS Teachers’ Adoption of Curriculum Innovations”. *Proceedings of the 40th ACM Technical Symposium on Computer Science Education. SIGCSE '09*. New York, NY, USA: ACM, 2009, pp. 544–548.
- [NM08] Nichols, A. L. & Maner, J. K. “The good-subject effect: Investigating participant demand characteristics”. *The Journal of general psychology* **135.2** (2008), pp. 151–166.
- [PV13] Paul, W. & Vahrenhold, J. “Hunting high and low: Instruments to detect misconceptions related to algorithms and data structures”. *Proceeding of the 44th ACM technical symposium on Computer science education*. ACM. 2013, pp. 29–34.
- [Pil03] Pillay, N. “Developing intelligent programming tutors for novice programmers”. *ACM SIGCSE Bulletin* **35.2** (2003), pp. 78–82.
- [Por13] Porter, L. et al. “Evaluating student understanding of core concepts in computer architecture”. *Proceedings of the 18th ACM conference on Innovation and technology in computer science education*. ACM. 2013, pp. 279–284.
- [Pot04] Potter, M. et al. “The Nominal Group Technique: A useful consensus methodology in physiotherapy research.” *New Zealand Journal of Physiotherapy* **32.3** (2004).
- [Pow03] Powell, C. “The Delphi technique: myths and realities”. *Journal of advanced nursing* **41.4** (2003), pp. 376–382.
- [Pri15] Price, T. W. et al. “BJC in Action : Comparison of Student Perceptions of a Computer Science Principles Course”. *Proc. of the 1st Annual RESPECT Conference*. 2015.

- [Pri16] Price, T. W. et al. "Lessons Learned from "BJC" CS Principles Professional Development". *Proc. 47th ACM Tech. Symp. on CS Ed. SIGCSE '16*. Memphis, Tennessee: ACM, 2016, pp. 467–472.
- [RW99] Rowe, G. & Wright, G. "The Delphi technique as a forecasting tool: issues and analysis". *International journal of forecasting* **15.4** (1999), pp. 353–375.
- [Sac74] Sackman, H. *Delphi assessment: Expert opinion, forecasting, and group process*. Tech. rep. RAND CORP SANTA MONICA CA, 1974.
- [SH14] Schmidt, F. L. & Hunter, J. E. *Methods of meta-analysis: Correcting error and bias in research findings*. Sage publications, 2014.
- [Sch08] Schoenberg, J et al. "Change it up! What girls say about redefining leadership". *New York, NY: Girl Scouts of the USA* (2008).
- [Shu70] Shulman, L. S. "Reconstruction of educational research". *Review of Educational Research* **40.3** (1970), pp. 371–396.
- [Shu86] Shulman, L. S. "Those Who Understand: Knowledge Growth in Teaching". English. *Educational Researcher* **15.2** (1986), pp. 4–14.
- [Sku07] Skulmoski, G. J. et al. "The Delphi method for graduate research". *Journal of Inf. Tech. Ed* **6** (2007), p. 1.
- [Sny12] Snyder, L. et al. "Computer Science Principles: an analysis". *ACM Inroads* **3.2** (2012), pp. 69–71.
- [Ste14] Stegeman, M. et al. "Towards an Empirically Validated Model for Assessment of Code Quality". *Proceedings of the 14th Koli Calling International Conference on Computing Education Research*. Koli Calling '14. New York, NY, USA: ACM, 2014, pp. 99–108.
- [Ste16] Stegeman, M. et al. "Designing a Rubric for Feedback on Code Quality in Programming Courses". *Proceedings of the 16th Koli Calling International Conference on Computing Education Research*. Koli Calling '16. New York, NY, USA: ACM, 2016, pp. 160–164.
- [Ste04] Stemler, S. E. "A comparison of consensus, consistency, and measurement approaches to estimating interrater reliability". *Practical Assessment, Research & Evaluation* **9.4** (2004), pp. 1–19.
- [SL05] Stevens, D. D. & Levi, A. J. "Introduction to rubrics". *Sterling, VA: Stylus* (2005).
- [SZ75] Strauss, H. J. & Zeigler, L. H. "The Delphi technique and its uses in social science research". *The Journal of Creative Behavior* **9.4** (1975), pp. 253–259.

- [SS14] Sung, K. & Snyder, L. “A Case of Computer Science Principles with Traditional Text-based Programming Languages”. *J. Comput. Sci. Coll.* **30.1** (2014), pp. 161–172.
- [Tay14] Taylor, C. et al. “Computer science concept inventories: past and future”. *Computer Science Education* **24.4** (2014), pp. 253–276.
- [Tei06] Teijlingen, E. van et al. “Delphi method and nominal group technique in family planning and reproductive health research”. *Journal of Family Planning and Reproductive Health Care* **32.4** (2006), pp. 249–252. eprint: <http://j.fprhc.bmj.com/content/32/4/249.full.pdf>.
- [TG10] Tew, A. E. & Guzdial, M. “Developing a Validated Assessment of Fundamental CS1 Concepts”. *Proceedings of the 41st ACM Technical Symposium on Computer Science Education. SIGCSE '10*. New York, NY, USA: ACM, 2010, pp. 97–101.
- [TG11] Tew, A. E. & Guzdial, M. “The FCS1: a language independent assessment of CS1 knowledge”. *Proceedings of the 42nd ACM technical symposium on Computer science education*. ACM. 2011, pp. 111–116.
- [TDC79] Thomas D. Cook, D. T. C. “Quasi-Experimentation: Design and Analysis Issues for Field Settings”. *Journal of Personality Assessment* (1979).
- [Tri15] Trinity College. *Mobile CS Principles*. web. url=[mobile-csp.org/resources](http://mobile-csp.org/resources). 2015.
- [Tuc03] Tucker, A. *A Model Curriculum for K–12 Computer Science: Final Report of the ACM K–12 Task Force Curriculum Committee*. Tech. rep. ACM Order No.: 104043. New York, NY, USA: ACM, 2003.
- [Tur70] Turoff, M. “The design of a policy Delphi”. *Technological forecasting and social change* **2.2** (1970), pp. 149–171.
- [Uhl02] Uhlenbeck, A. “The development of an assessment procedure for beginning teachers of English as a foreign language”. Doctoral Thesis. P.O. Box 9555, 2300 RB Leiden, The Netherlands: Leiden University, 2002.
- [Uni14] University of California, Berkeley. *Hangman Classic*. Online. URL@ <http://bjc.berkeley.edu/bjc-r/cur/programming/projects/hangman/hangman-classic.html>. 2014.
- [Uni15] University of California, Berkeley. *BJC - Beauty and Joy of Computing*. 2015. URL: [bjc.berkeley.edu](http://bjc.berkeley.edu).
- [Uni17] University of California, Berkeley. *Snap! Build Your Own Blocks*. 2017.
- [Ute] *UTeach CS Principles*. 2017. URL: <https://cs.utexas.edu/>.

- [WS74] Weatherman, R. & Swenson, K. “Delphi technique”. *Futurism in education: Methodologies* (1974), pp. 97–114.
- [WT14] Webb, K. C. & Taylor, C. “Developing a pre-and post-course concept inventory to gauge operating systems learning”. *Proceedings of the 45th ACM technical symposium on Computer science education*. ACM. 2014, pp. 103–108.
- [Wil11] Wilson, C. et al. *Running on empty*. New York, NY, 2011.
- [Win06] Wing, J. M. “Computational Thinking” (2006), pp. 33–35.
- [Win97] Winzenried, A. “Delphi Studies: The Value of Expert Opinion Bridging the Gap—Data to Knowledge.” (1997).
- [Wol11] Wolz, U. et al. “Scrape: A tool for visualizing the code of Scratch programs”. *Poster at the 42Nd ACM Technical Symposium on Computer Science Education*. SIGCSE ’11. Dallas, TX, USA: ACM, 2011.
- [Yad16] Yadav, A. et al. “Measuring Computer Science Pedagogical Content Knowledge: An Exploratory Analysis of Teaching Vignettes to Measure Teacher Knowledge”. *Proceedings of the 11th Workshop in Primary and Secondary Computing Education*. WiPSCE ’16. New York, NY: ACM, 2016, pp. 92–95.
- [Yua16] Yuan, A. et al. “Almost an Expert: The Effects of Rubrics and Expertise on Perceived Value of Crowdsourced Design Critiques”. *Proceedings of the 19th ACM Conference on Computer-Supported Cooperative Work & Social Computing*. CSCW ’16. New York, NY, USA: ACM, 2016, pp. 1005–1017.
- [Zel00] Zeller, A. “Making Students Read and Review Code”. *SIGCSE Bull.* **32.3** (2000), pp. 89–92.

## APPENDICES

## APPENDIX

# A

## RESOURCES

### **A.1 Beauty and Joy of Computing Lesson Plans**

pg. 132 Brick Wall Lab + Rubric

pg. 137 Hangman Lab + Rubric

pg. 140 Binary Conversion Lab + Rubric

pg. 143 C-Curve Lab + Rubric

pg. 148 eCard Lab + Rubric Training Survey

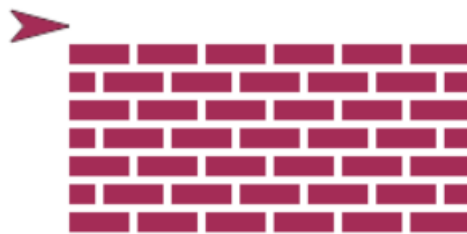
pg. 152 Shopping List Lab + Rubric Training Survey

pg. 157 Traffic Simulation Lab - No Rubric (No longer in circulation)

## LAB - BRICK WALL

### BRICK WALL

Sometimes, when we write programs and scripts, it feels like we have hit a brick wall. (This is a good sign; if a project isn't hard, you're not stretching your mind!) Now, you are going to draw this brick wall:



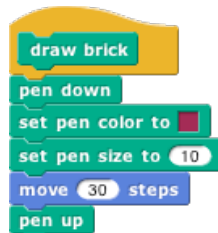
## Abstraction

While SNAP has lots of blocks for drawing and moving, it doesn't have anything about bricks. It wouldn't make sense to have one in general because most programs don't involve bricks. But ones that do could really use a `draw brick` block. `draw brick` moves us away from having to talk in the computer's terms and closer to using the terms of the problem we're trying to solve. The process of moving from what the programming language gives you to a natural language (one that humans speak to communicate ) is critical to a good abstraction.

In this problem, you will build an abstraction for drawing a brick wall, first by creating `draw brick`, then blocks for drawing rows, and ultimately the goal: `Draw a Brick Wall with Rows`.

## Drawing One Brick

A brick is a solid red rectangle ("brick red," i.e., not a bright primary-color red but somewhat darker). There's no "draw rectangle" block in SNAP, but we can fake it by thinking of a rectangle as a very thick line. You know how to draw a line, with the **move** block. In the **Pen** palette there's a **set pen size** block. We can use it to draw a thick line:



If you try out this block, you'll notice that your brick has rounded ends. These ends stick out beyond the length of the line you asked to draw. Here's a picture of the rounded brick with a regular line (pen size 1) inside it:



These rounded ends are very pretty when drawing most pictures, but they don't look like bricks, and they make thinking about lengths in this project too complicated. So for this project you're going to turn off the rounded ends. Click on the settings button (⚙️) in the toolbar, and turn on the "**Flat line ends**" setting.

## Using Problem Decomposition to Write an Abstraction

Consider this line of code that was used to create the brick wall:



If you were to ask a mason to make a brick wall with seven rows, he would surely understand your meaning and make it happen. A computer, however, doesn't know what that means, so you have to fill in the details. This means going from the abstract (draw a brick wall) to the concrete (pardon the pun), which involves problem decomposition.

A quick observation shows that there are two kinds of rows of bricks:

- Row A: 
- Row B: 

Make blocks `rowA` and `rowB`. Think about what helper blocks besides `draw brick` you might want.

It's possible to go overboard on abstraction, so that you have a gazillion blocks and it's hard to find where the work actually gets done. But, on the other hand, sometimes it's useful to make a custom block even if its definition is just one primitive block. For example, to draw the *mortar* (the white space) between blocks in a row, all you have to do (since the `draw brick` block picks up the pen at the end of its script) is move forward:

`move 4 steps`

You could just use that `move` block inside your **Row A** block. But you might instead want to define a **Draw Mortar** block. Why? Maybe later you'll decide that four steps is the wrong thickness for mortar, and you'd rather have five steps. If there are a dozen **Move** blocks scattered through your program to draw mortar, you might not find them all. With a **Draw Mortar** block, you can just change its definition, and all the mortar in your picture will be changed.

Notice that the two kinds of rows should be exactly the same length. Your first try at drawing a Row B will probably be a little too long. Why? Row A has six whole bricks. Row B has five whole bricks plus two half-bricks, which adds up to six whole bricks. To understand the bug, think about the amount of *mortar* in each kind of row.

How are you going to fix it? Should a Row B have different-size bricks, different-size mortar gaps, or different-size half-bricks? If you're not sure, try all the possibilities and see which looks right in the finished wall. (There's really only one good answer.)

Once you have the two kinds of rows the same length, you can write the `Draw a Brick Wall with  Rows` block. Remember that it should work for both even and odd numbers of rows, which means that sometimes you'll have to draw an extra Row A.

(How do you know if a number is odd? You'll find the  `mod`  block helpful.)

## Additional Exercises

After you've drawn the picture at the top of this page, add more inputs to the **Draw a Brick Wall** block:

- Length of the wall (how many bricks)
- Length of a brick
- Mortar thickness

Add these one at a time, not all at once! When you modify the length of a brick, that should also change the length of a half-brick. When you modify the mortar thickness, that should also change the distance between rows (since that's mortar too).

Game: Making a Brick Wall

Student Name: \_\_\_\_\_

Category/Points	4	3	2	1	0	Learning Objective/Essential Knowledge
Abstraction	Code properly separated into deeper functions and abstractions. Different abstractions for each process: Draw brick Rows Draw Brick Wall Draw Mortar	Majority of code is separated into different levels of abstraction with few functions not divided into new blocks for functionality.	Some of the code is abstracted properly, using different functions and code blocks to perform basic actions. Few functions are left non abstracted.	Little of the code is properly abstracted with the majority of functions left together.	Code all contained within one singular block, not separated into more blocks and functions.	EK 2.2.1B An abstraction extracts common features from specific examples in order to generalize concepts. EK 5.3.1A Procedures are reusable programming abstractions.
Visual	Brick Wall is styled correctly: Brick Color Sharp Edges Consistent size/shape Consecutive layers are offset. Layers are offset properly with use of whitespace.	Meets the majority of the styling requirements.	Meets half of the styling requirements.	Meets few of the styling requirements.	Meets none of the required styling conditions.	EK 2.3.1D Simulations mimic real-world events without the cost or danger of building and testing the phenomena in the real world
Mathematics	Code uses mathematics (loops, mod, etc.) to automatically monitor when the code begins, ends, and how it processes.	Majority of code uses mathematics to monitor and keep track of how it runs.	Some mathematics is used to automate the functions but there are uses of numerous manual function calls that could be replaced with math to automate it.	Little mathematics is used to automate the process of drawing the wall and the program relies on manual function block calls.	Uses no mathematics to monitor the beginning/end of the process of drawing and contains numerous cases of same function call.	EK 4.1.1D Iteration is the repetition of part of an algorithm until a condition is met or for a specified number of times.
Use of Parameters	Code blocks and abstract functions use parameters to increase usability and are easily changable for testing.	Majority of abstractions utilize parameters to be used for testing.	Some functions and abstractions utilize parameters to allow greater functionality.	Few abstractions utilize parameters.	Abstractions do not utilize parameters.	EK 5.3.1F Parameters generalize a solution by allowing a procedure to be used instead of duplicated code.
Style/Layout	Code cleanly organized and laid out to be read easily. Abstractions/ blocks are properly composed.	Much of the code is cleanly organized with certain parts disjointed/apart.	Half of the code is cleanly organized with the other half disjointed and separated.	Majority of code is broken apart and hard to follow by the reader.	Code is laid out sporatically and cannot easily be followed by the reader/grader.	EK 3.1.3A Visualization tools and software can communicate information about data.
Naming	Naming of functions, including abstractions, accurately represents the problem being solved.		Some functions named properly, others provide no meaning.		All functions named improperly, not representing the problem at hand.	EK 5.4.1C Meaningful names for variables and procedures help people better understand programs. EK 5.1.2E Documentation about program components, such as blocks and procedures, helps in developing and maintaining programs.
Collaboration (Optional)				Classmates work with assigned team mate and complete/turn in project together.	Classmates did not work together.	EK 3.1.2A Collaboration is an important part of solving data-driven problems.

## MORE ON COMPOSITION OF HIGHER ORDER FUNCTIONS



This is the corrected version of **acronym** that checks for words starting with a capital letter. Isn't it beautiful? It does a complicated job, so there's a lot packed in there, but think how much worse it would be if we didn't have lists to help organize such tasks. You'd take the text string **phrase** and write a loop to go through it, character by character, looking for spaces as word separators. Then you'd have to build up the result string, adding one letter at a time.

With **map**, **keep**, and **combine**, you can operate on the items of a list *all at once*. You don't have to think, "Find the first letter of the first word and operate on it, then increase the loop index until you find a space, then skip over any extra spaces that might be next to it, then remember the position of the beginning of the second word" and so on. You can think, "give me the first letters of all the words."

(Instead, all that ugliness about looking for spaces is hidden inside the **sentence->list** block, where everyone writing an application about sentences can use it without having to reinvent it. This is another example of *abstraction*, which we mentioned

right at the beginning as one of the central ideas of the course and of computer science in general.)

### Try this:

- Write a **max** block that takes two numbers as inputs and reports the bigger one (either of them, if they're equal). Use it, and the list tools, to find the length of the longest word of a sentence.
- Using that length to help, write a block that reports the longest word of a sentence (or the first word of that length, if there's a tie).
- Write a **word->list** block that takes a word of text as input, and reports a list in which each item is a single letter from the word. To do this, you'll have to use a loop, along with the **add to list** block:



- Imagine that you're writing a program to play Hangman. The program has thought of a secret word, and the user is trying to guess it. Write a **display word** block that takes two inputs, the secret word and a list of the letters guessed by the user so far. It should **say** the letters of the secret word, spaced out, with underscore characters replacing the letters not yet guessed:



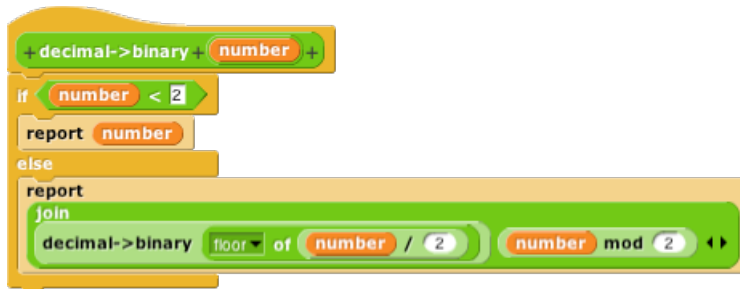
(Use your **word->list** block on the secret word to get started.)

Student Name: \_\_\_\_\_

Category/Points	4	3	2	1	0	Learning Objective/Essential Knowledge
Abstraction	Code properly separated into deeper functions and abstractions. Different abstractions for each process: Word-List Display Word (etc.)	Majority of code is separated into different levels of abstraction with few functions not divided into new blocks for functionality.	Some of the code is abstracted properly, using different functions and code blocks to perform basic actions. Few functions are left non abstracted.	Little of the code is properly abstracted with the majority of functions left together.	Code all contained within one singular block, not separated into more blocks and functions.	EK 2.2.1B An abstraction extracts common features from specific examples in order to generalize concepts.  EK 5.3.1A Procedures are reusable programming abstractions.
Algorithm Composition	Code cleanly organized and laid out to be read easily. Abstractions/blocks are properly composed.	Much of the code is cleanly organized with certain parts disjointed/apart.	Half of the code is cleanly organized with the other half disjointed and separated.	Majority of code is broken apart and hard to follow by the reader.	Code is laid out sporatically and cannot easily be followed by the reader/grader.	LO 4.1.2 Express an algorithm in a language.  LO 4.4.1 Develop an algorithm for implementation in a program.
Mathematics	Code uses math and logic (loops, mod, etc.) to automatically monitor when the code begins, ends, and how it processes.	Majority of code uses mathematics to monitor and keep track of how it runs.	Some mathematics is used to automate the functions but there are uses of numerous manual function calls that could be replaced with logic to automate it.	Little or incorrect math is used to automate the program process and the program relies on manual function block calls.	Uses no mathematics to monitor the beginning/end of the process of drawing and contains numerous cases of same function call.	EK 5.5.1D Mathematical expressions using arithmetic operators are part of most programming languages.
Use of List/Parameters	Code blocks and abstract functions use list operators & parameters to increase program efficiency.	Majority of abstractions utilize parameters to be used for testing, some list operators are used for efficiency.	Some functions and abstractions utilize parameters and list operators are not used efficiently.	Few abstractions utilize parameters.	Abstractions do not utilize parameters.	EK 5.3.1F Parameters generalize a solution by allowing a procedure to be used instead of duplicated code.  EK 5.3.1L Using lists and procedures as abstractions in programming can result in programs that are easier to develop and maintain.
Correctness/Readability	Naming of functions, including abstractions, accurately represents the problem being solved program is easy to read; program works.		Some functions named properly, others provide no meaning, program can be understood after a few read throughs, and mostly works		All functions named haphazardly, difficult to understand, or doesn't work	LO 5.4.1 Evaluate the correctness of a program.  EK 5.4.1C Meaningful names for variables and procedures help people better understand programs.

## BEYOND BINARY

Here's our solution:

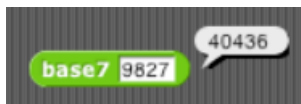


The base case is that the number fits in a single bit—that is, it has to be less than 2. If so, the number itself, 0 or 1, is the desired output.

In the recursive case, the rightmost bit of the result is the remainder of dividing the number by 2. That is, even numbers end with 0, and odd numbers end with 1. The rest of the result is a recursive call on the (integer) quotient of the number divided by 2. The combiner is **join** because we want to string the digits together. It may be surprising that we don't use an arithmetic operator, since we're working with numbers, but the desired result is a **numeral**, which is a visible representation of a number, rather than the numeric value itself. A numeral is a text string, so the combiner is a string operation.

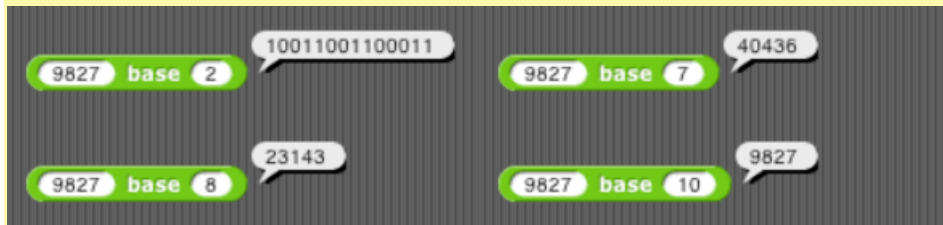
## Other Bases

There's no reason to limit ourselves to decimal (base 10) and binary (base 2). How about base 7?

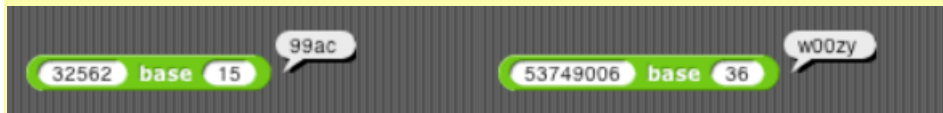


The base 7 digits are 0–6, and the digit positions represent powers of 7.

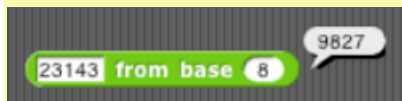
👉 Write the **base7** block. Then generalize the pattern with a **base** block that takes the base as a second input:



👉 Optional for hotshots: Improve the **base** block so that it can go up to base 36 by using the letters **a–z** as digits with values 10–35.



👉 Write the inverse function **from base** that takes a (text) string of digits and a base as inputs, and reports the corresponding number (which Snap! will show in decimal, of course, but you're converting to a **number**, not to a string of digits). Again, bases above ten are optional.



## Binary Numbers

Student Name: \_\_\_\_\_

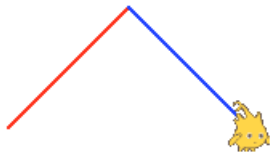
CATEGORY	4	3	2	1	0	Learning Objective/Essential Knowledge
<b>Abstraction</b>	Code properly separated into multiple abstractions. Should have separate blocks defined (i.e. pascal, next row - or the equivalent functions).	Code separated into multiple abstractions with only minor overlapping between functions.	Code separated into multiple abstractions with function that do the same thing.	Little of the code is properly abstracted with the majority of functions left together.	Code all contained within one singular block, not separated into more blocks and functions.	EK 2.2.1C An abstraction generalizes functionality with input parameters that allow software reuse.
<b>Visual</b>	Reports a strings of zeros and ones is that correctly converts to binary from decimal (user input).		Reports a strings of zeros and ones is that attempts to converts to binary from decimal (user input).		Nothing is reported	EK 3.1.3A Visualization tools and software can communicate information about data.
<b>Mathematics</b>	Code uses a combination of mathematical concepts (floor, mod, etc) to calculate the binary numbers. Mathematics is used to automate function calls.	Majority of the code uses a combination of mathematical concepts (floor, mod, etc) to calculate the binary numbers. Some hardcoded base cases are with in the math. Mathematics is used to automate function calls.	Some of the code uses a combination of mathematical concepts (floor, mod, etc) to calculate the binary numbers. Some hardcoded base cases are with in the math. Minimal mathematics is used to automate function calls.	Most of the code has hardcoded conversions from decimal to binary instead of calculating using operations, such as floor or mod. Function calls are not automated.	Uses no mathematics to monitor the loops.	EK 2.1.1G Numbers can be converted from any base to any other base. EK 4.1.1D Iteration is the repetition of part of an algorithm until a condition is met or for a specified number of times. EK 5.5.1A Numbers and numerical concepts are fundamental to programming.
<b>Use of Parameters</b>	Code blocks and abstract functions use parameters to increase usability and are easily changable for testing. Parameters allow user to input a number that is decimal (base 10).	Majority of abstractions utilize parameters to be used for testing.	Some functions and abstractions utilize parameters to allow greater functionality.	Few abstractions utilize parameters.	Abstractions do not utilize parameters. Do no use parameters from user input.	EK 5.3.1G Parameters provide different values as input to procedures when they are called in a program.
<b>Style/Layout</b>	Code cleanly organized and laid out to be read easily. Abstractions/ blocks are properly composed.	Much of the code is cleanly organized with certain parts disjointed/apart.	Half of the code is cleanly organized with the other half disjointed and separated.	Majority of code is broken apart and hard to follow by the reader.	Code is laid out sporatically and cannot easily be followed by the reader/grader.	EK 5.4.1A Program style can affect the determination of program correctness. EK 5.4.1B Duplicated code can make it harder to reason about a program.
<b>Naming</b>	Naming of functions, including abstractions, accurately represents the problem being solved.		Some functions named properly, others provide no meaning.		All functions named improperly, not representing the problem at hand.	EK 5.4.1C Meaningful names for variables and procedures help people better understand programs. EK 5.1.2E Documentation about program components, such as blocks and procedures, helps in developing and maintaining programs.

## C-CURVE

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. Below the base case is that the sprite draws a single line. The sprite starts facing right and faces right at the end. (Hint - the direction that the sprite points at the end is important! It should point in the same direction it did at the beginning of the recursive call.)



In the next level, start facing right and end facing right but repeat the previous level twice (red and blue below).

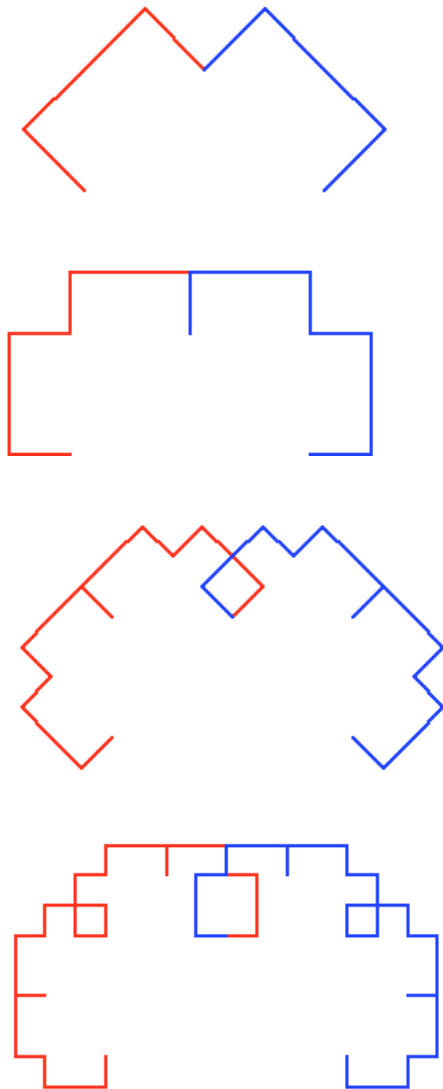


In the next level, start facing right and end facing right but repeat the previous level twice (red and blue below).

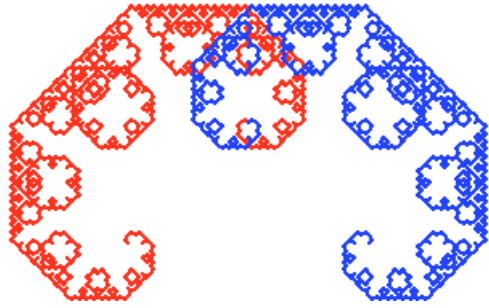


In the next level, start facing right and end facing right but repeat the previous level twice (red and blue below). This continues for each of the following levels, but you should probably focus on the trying to think through the first few levels.

If it's helpful to you, use the same technique we showed in writing the tree program: Start with a **flake1** block that handles the base case, then write a **flake2** block, then **flake3**, and so on until you understand the pattern.







### Fractals: C-Curve Fractal

Student Name: \_\_\_\_\_

CATEGORY	4	3	2	1	0	Learning Objective/Essential Knowledge
<b>Abstraction</b>	Code properly separated into multiple abstractions. There should be a hierarchy of functions (the main C-Curve fractal level should call other functions).	Code separated into multiple abstractions with only minor overlapping between functions.	Code separated into multiple abstractions with function that do the same thing.	Little of the code is properly abstracted with the majority of functions left together.	Code all contained within one singular block, not separated into more blocks and functions.	EK 2.2.1C An sbtraction generalizes functionality with input parameters that allow software.
<b>Visual</b>	C-Curve are styled correctly: - see image online - variable size - variable number of levels. Levels are correctly added to the corners of the main center C-Curce (angles all correct)	C-Curve are styled correctly: - see image online - variable size - variable number of levels	C-Curve are styled as: - see image online There is either variable number of levels or variable size.	C-Curve are styled as: - see image online There is not variable levels or variable size	Initial C-Curve does not appear as initial C-Curve online.	EK 3.1.3A Visualization tools and software can communicate information about data.
<b>Mathematics</b>	Code uses mathematics (loops, operations, ...) to monitor length of C-Curve, how many levels of C-Curve to make.	Majority of code uses mathematics to monitor either length of triangles or how many C-Curve to make.	Some mathematics is used to automate the functions but there are uses of numerous manual function calls that could be replaced with math to automate it.	Little mathematics is used to automate the process of drawing the treesl and the program relies on manual function block calls.	Uses no mathematics to monitor the loops for drawing the trees and contains numerous cases of same fuction call instead of using loops or calculated numbers.	EK 4.1.1E Algorithms can be combined to make new algorithms. EK 4.1.1F Using existing correct algorithms as building blocks for constructing a new algorithm helps ensure the new algorithm is correct. EK 5.5.1A Numbers and numerical concepts are fundamental to programming.
<b>Use of Parameters</b>	Code blocks and abstract functions use parameters to increase usability and are easily changable for testing. Parameters allow user to input size of C-Curve and number of levels.	Majority of abstractions utilize parameters to be used for testing. Most parameters are from user inputs.	Some functions and abstractions utilize parameters to allow greater functionality. Some parameters are from user inputs.	Few abstractions utilize parameters. Do no use parameters from user input.	Abstractions do not utilize parameters.	EK 5.3.1G Parameters provide different values as input to procedures when they are called in a program.
<b>Style/Layout</b>	Code cleanly organized and laid out to be read easily. Abstractions/ blocks are properly composed.	Much of the code is cleanly organized with certain parts disjointed/apart.	Half of the code is cleanly organized with the other half disjointed and separated.	Majority of code is broken apart and hard to follow by the reader.	Code is laid out sporatically and cannot easily be followed by the reader/grader.	EK 5.4.1A Program style can affect the determination of program correctness. EK 5.4.1B Duplicated code can make it harder to reason about a program.
<b>Naming</b>	Naming of functions, including abstractions, acurately represents the problem being solved.		Some functions named properly, others provide no meaning.		All functions named improperly, not representing the problem at hand.	EK 5.4.1C Meaningful names for variables and procedures help people better understand programs. EK 5.1.2E Documentation about program components, such as blocks and procedures, helps in developing and maintaining programs.



# eCard Activity

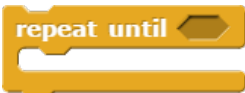
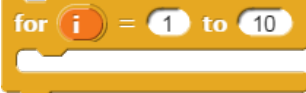
---

## For You To Do

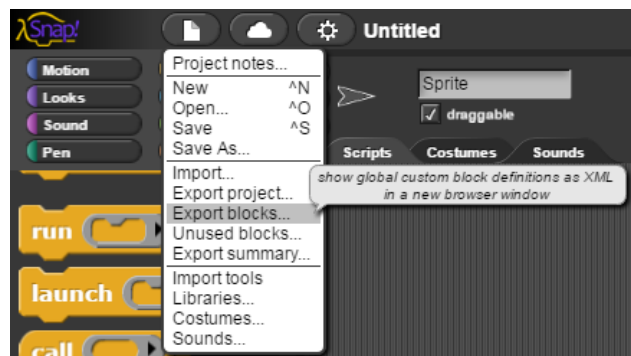
Create an eCard that meets **all** of the following specifications:

1. Has a message for the viewer (such as "Happy New Year," "Happy Birthday," "Get Well Soon," etc.)
2. Has stage and sprite costumes
3. Works properly (is not buggy)
4. Includes a custom block. This can be *either*:
  - A **start over** block that resets the eCard to the beginning (resets pen, stage, and sprites) OR
  - Another custom block containing at least 3 blocks (this could move a sprite, change the background, etc.)

5. Uses  or  to customize the card based on some information about the recipient (e.g., age or gender)

6. Uses  or , for example, to animate a sprite or draw a pattern in the background.
7. Lets the user interact (e.g., by clicking a sprite, pressing a key, or moving the mouse)

When you have finished your app, export it to an .xml file (see image below), upload it to Piazza, and write a short description of your program. If necessary, also say how to use it.



## BJC 2017 - eCard Grading Rubric

Please use the rubric below to rate your partner's project. Then upload the project file below.

### Category: Creativity

*Learning Objective: Create a computational artifact for creative expression.*

Programs developed for creative expression, to satisfy personal curiosity, or to create new knowledge may have visual, audible, or tactile inputs and outputs.

Select the rating which best applies to the project:

- 0: An e-card has not been created
- 1: A message has been displayed or costumes have been utilized
- 2: A message is displayed on a costumed stage
- 3: The e-card features a message with both stage and sprite costumes
- 4: The user can interact with the e-card to cause a change to creative elements of the card

### Category: Algorithms/Loops

*Learning Objective: Develop an algorithm for implementation in a program.*

Iteration is the repetition of part of an algorithm until a condition is met or for a specified number of times.

Select the rating which best applies to the project:

- 0: No loops or iteration was used
- 1: The program implements a loop incorrectly
- 2: The program implements a loop correctly
- 3: The program correctly uses a repeat until or for block to visually change the e-card
- 4: The program correctly uses at least one repeat until or for blocks to contribute to the creativity of the e-card

### Category: Abstraction

*Learning Objective: Use abstraction to manage complexity in programs.*

Duplicate Code and longer code segments are harder to reason about than shorter code segments in a program.

Select the rating which best applies to the project:

- 0: No custom block was used
- 1: The program implements a custom block
- 2: The program reuses a custom block
- 3: The program uses a custom block containing at least three other blocks
- 4: The program uses custom blocks to help segment the program into reasonable chunks

**Category: Logic**

*Learning Objective: Employ appropriate mathematical and logical concepts in programming.*

Logical concepts and Boolean algebra are fundamental to programming.

Select the rating which best applies to the project:

- 0: No conditionals were used
- 1: A conditional statement is implemented incorrectly
- 2: The program properly utilizes conditional logic
- 3: The program uses conditional logic to customize the e-card
- 4: The program uses conditional logic to customize the e-card based on user input

**Category: Documentation**

*Learning Objective: Documentation helps in developing and maintaining programs when working individually or in collaborative programming environments.*

Meaningful names for variables and procedures help people better understand programs.

Select the rating which best applies to the project:

- 0: Code is not documented and hard to interpret
- 1: Code is not documented or named well
- 2: Code is named aptly, but difficult to interpret
- 3: Variables and methods are named so that code is easy to interpret
- 4: Code is well documented with comments and easy to interpret

**Category: Writing**

*Learning Objective: Use written language to explain how a program meets its specifications.*

An explanation of a program helps people understand the functionality and purpose of it and is often described by how a user interacts with it.

Select the rating which best applies to the project:

- 0: No written statement included
- 1: Written statement does not describe program
- 2: Written statement gives vague understanding of program
- 3: Written statement adequately explains the program
- 4: Written statement uses language to give a high-level overview on the functionality of the program and how users interact with it

Please upload the project file you just graded. (It will look like "Project-Name.xml").

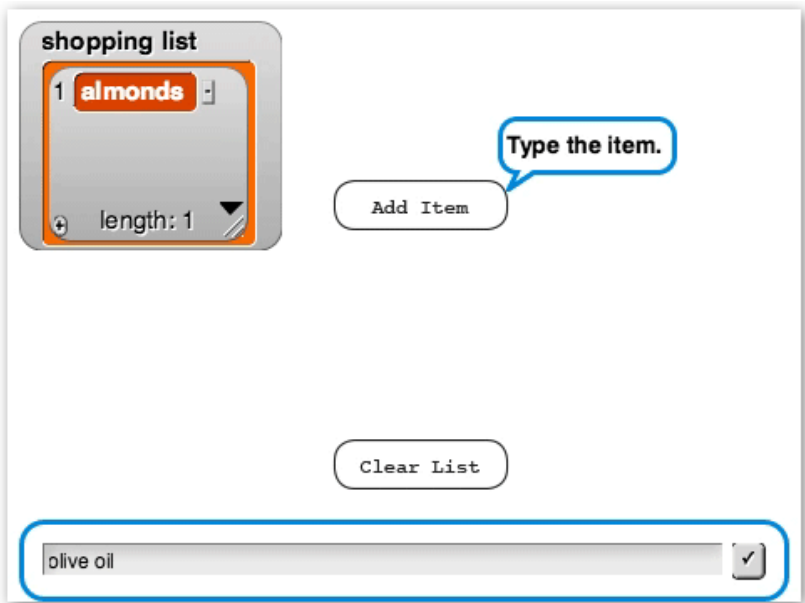
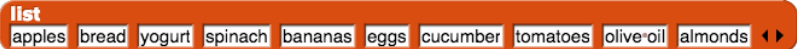
Powered by Qualtrics

# Shopping List App

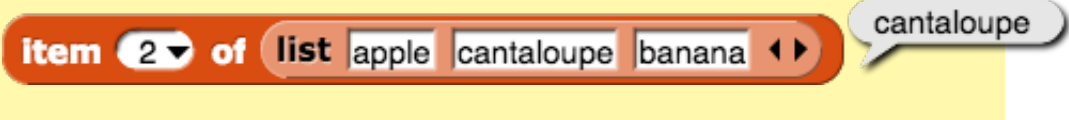


Remember to Do Pair Programming Swaps

Many computer apps—contact lists, playlists, calendars of events, locations on a map, reminders—involve manipulating *lists* of information with tools that search, sort, or change the items on the list. In this first activity, you'll create a Shopping List app.







You've worked with lists before. In this unit you will learn more about this powerful data structure. Lists can contain letters or words, as you've seen before, or lists, as you'll see in this unit, or even blocks, as you'll see in Unit 6. When you used **map** with lists in Unit 1, it worked with *all* the items in a list. You can also select a specific item in a list by specifying its *position* by number. That number is called the **index**.



## For You To Do

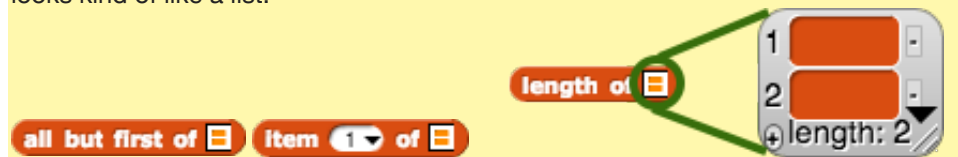
1. [Load this project.](#)

2.  called `shopping list` to store the information.

3. Use  to set the starting value of `shopping list` to be an empty list. You will need the list block . Use its arrows to get rid of its input slot so that it looks like this: .

Setting the starting value of a variable is known as **initializing** the variable.

Some block input slots expect a list as input. You can tell because the input slot looks kind of like a list:



4. Write a script for the "Add Item" button so that when that sprite is clicked, it will `ask` the user for a new item, and then add the new item to the grocery list. You can use the `add` or `insert` list block to modify the list. Make sure it works.

You first saw the `ask` and `answer` blocks in Unit 2: Using Multiple `If` Statements and Developing a Number Guessing Game.

5. Write a script for the "Clear List" button that asks for confirmation and then sets the variable back to an empty list.



Now is a good time to save your work...

6. Improve the "Add Item" button to add new items only if they are not already on the list. You can use `contains` to see if an item is already on the list.

7. Write a script for the "Search" button that tells the user whether an item is already on the list (but doesn't add the item).

8. Write a script for the "Delete Item" button that removes an item from the list.

9. Think of another improvement and program it.

## BJC 2017 - Shopping List Grading Rubric

Please use the rubric below to rate your partner's project. Then upload the project file below.

### Category: Creativity

*Learning Objective: Create a new computational artifact by combining or modifying existing artifacts.*

Programs developed for creative expression, to satisfy personal curiosity, or to create new knowledge may have visual, audible, or tactile inputs and outputs.

Select the rating which best applies to the project:

- 0: The starter code has not been modified
- 1: The starter code has been modified, but neither input nor creative aspects are used in the program
- 2: The starter program takes in user input or has an extra improvement
- 3: The starter program takes in user input and has been improved
- 4: User input has been added throughout the starter program and creative improvements have been implemented

### Category: Algorithms

*Learning Objective: Develop an algorithm for implementation in a program.*

Knowledge of standard algorithms can help in constructing new algorithms.

Select the rating which best applies to the project:

- 0: The add item button is not implemented
- 1: The add item button prompts the user when clicked
- 2: The add item button prompts and adds an item to a shopping list
- 3: The add item button checks if an item is on the shopping list
- 4: The add item button adds items only if they are not already on the shopping list

### Category: List

*Learning Objective: Use lists and procedures as abstractions to result in programs that are easier to maintain.*

Lists and list operations, such as add, remove, and search, are common in many programs.

Select the rating which best applies to the project:

- 0: No lists were used
- 1: The program implements an empty list
- 2: The program implements a list that stores items
- 3: Users can add items to the shopping list and clear it
- 4: Users can add, delete, and search for items on the shopping list

**Category: Logic**

*Learning Objective: Employ appropriate mathematical and logical concepts in programming.*

Selection uses a Boolean condition to determine which of two parts of an algorithm is used.

Select the rating which best applies to the project:

- 0: No conditionals were used
- 1: A conditional statement is present in the program
- 2: Conditional statements were used but did not utilize the contains block
- 3: The program uses conditional logic (incorrectly) to determine when an item should be added to the list
- 4: Conditionals are used with list operations (contains) to properly decide when items are added to the shopping list

**Category: Algorithms/Loops**

*Learning Objective: Develop an algorithm for implementation in a program.*

Iteration is the repetition of part of an algorithm until a condition is met or for a specified number of times.

Select the rating which best applies to the project:

- 0: No loops or iteration was used
- 1: The program implements a loop incorrectly
- 2: The program implements a loop correctly
- 3: The program correctly uses a loop to let users search for or delete items in the list

- 4: The program correctly uses multiple loops to complete or enhance their program

**Category: Documentation**

*Learning Objective: Develop a correct program to solve problems.*

Documentation and meaningful names for variables and procedures help people better understand programs.

Select the rating which best applies to the project:

- 0: Code is not documented and hard to interpret
- 1: Code is not documented
- 2: Code is named aptly, but difficult to interpret
- 3: Variables and methods are named so that code is easy to interpret
- 4: Code is well documented with comments and easy to interpret

Please upload the project file you just graded. (It will look like "Project-Name.xml").

Powered by Qualtrics









## APPENDIX

### B

# SURVEY INSTRUMENTS

## B.1 Study 3 (Delphi) Consent forms and Survey instruments

- pg. 162 National Brick Wall Delphi - Round 1
- pg. 166 National Brick Wall Delphi - Round 2
- pg. 169 National Brick Wall Delphi - Round 3
- pg. 179 Localized Hangman Delphi - Round 1
- pg. 183 Localized Hangman Delphi - Round 2
- pg. 186 Localized Hangman Delphi - Round 3





















































## B.2 Study 5 (Rubric Use) Consent forms and Survey instruments

pg. 189 BJC Rubric Evaluation Pre Survey - CSP Teachers

pg. 196 BJC Rubric Evaluation Pre Survey - STEM Education Majors (Similar to Previous)

pg. 203 Binary Conversion Grading Activity - Sample 1

pg. 207 Binary Conversion Grading Activity - Sample 2

pg. 211 Binary Conversion Grading Activity - Sample 3

pg. 215 BJC Rubric Evaluation Post Survey - CSP Teachers

pg. 250 BJC Rubric Evaluation Post Survey - STEM Education Majors (Similar to Previous)





























































































































































































7\$HZ LZ \$SJ[ [CL YLNUZ[CH] RÅ\ LWLK` V YKLJZ VU[V T HRL [CL Y[RN MY : [ ` \$ 3H V[

7\$HZ LZ \$SJ[ [CL YLNUZ[CH] RÅ\ LWLK` V YKLJZ VU[V T HRL [CL Y[RN MY 5 HT RN

) S/JR

/ V^ KIK` V ÄUK V [ H V [ [OZ Z Y]L` &

- ; OX NOHT HEN SZ
- VY HKLK I` HMLUK VYJVSHML
- 9LKH7VZ [R]HKEZ J Z Z/ØWY T

; VCLSM Z SUR` V YZ Y]L` [V RUMYT H [RU WXY] RKL R [CL WYL Z Y]L` 7\$HZ LY L U]LY` V Y  
LT HSHKKYLZ Z

Participants may also receive up to an additional \$5 by providing the name and contact information for a STEM education major or teacher who will be invited to complete the study. °

7\$HZ LZ MWH[L°LHJOLT HSHKKYLZ Z° POHZ Ø PJVSU "

: \NNLZ [K, T HSHKKYLZ ZZ[V 9LJY P