

USING KNOWLEDGE ELICITATION TECHNIQUES TO ESTABLISH A BASELINE OF
QUANTITATIVE MEASURES OF COMPUTATIONAL THINKING SKILL ACQUISITION
AMONG UNIVERSITY COMPUTER SCIENCE STUDENTS

A Dissertation

Presented to

The Faculty of the Department of Library Science and Technology

Sam Houston State University

In Partial Fulfillment

of the Requirements for the Degree of

Doctor of Education

by

M. Earnest Morrow

December, 2019

USING KNOWLEDGE ELICITATION TECHNIQUES TO ESTABLISH A BASELINE OF
QUANTITATIVE MEASURES OF COMPUTATIONAL THINKING SKILL ACQUISITION
AMONG UNIVERSITY COMPUTER SCIENCE STUDENTS

by

M. Earnest Morrow

APPROVED:

Li-Jen Lester, EdD
Dissertation Co-Director

Rebecca Robles-Piña, PhD
Dissertation Co-Director

Donggil Song, PhD
Committee Member

Gary W. Smith, PhD
Committee Member

Stacey L. Edmonson, PhD
Dean, College of Education

DEDICATION

I have heard it said that teachers change the world one student at a time. Of course, that does not imply that all students have the course of their lives altered by each of their teachers. My observation suggests that such impactful experiences are rare. An adult can consider themselves fortunate to have had one such encounter in their past: one mentor who changed their life.

This paper is dedicated to the three teachers most responsible for the life changing interventions that have brought me to this point. These individuals, along with my wife Gracie, allowed me to express my dreams, encouraged me to pursue them, and enable my accomplishments.

At a time before even pocket calculators were available, Carol Seljeseth, was the middle school science teacher who took an active interest in my passion to learn about computers. When I told her I was designing circuits for binary arithmetic but lacked access to the hardware to build and test them, she talked Honeywell into sponsoring me and providing an engineer as a tutor. The following summer she enrolled both of us in a programming class at the University of Oklahoma, where the path to my professional career in information technology was set.

Ten years ago I retired from that professional career. I wondered if I could apply my experience and knowledge to the field of Education. A little research led me to SHSU, and with some trepidation I decided to approach a professor about my desire. Dr. Li-Jen Lester welcomed me, unscheduled, into her office and listened to my ideas and addressed my concerns. She assured me that there was a role for me in Education, and encouraged me to begin my graduate studies. She also served for me as the role model of

a professional able to integrate computer science and Education. I thank Dr. Lester for always being willing to let me express my aspirations and for her constant encouragement over these past ten years.

Dr. Marilyn Rice has also always been very welcoming of my unscheduled stops by her office. For the first few years we discussed her vision for a doctorate program that would integrate technology and education. Her efforts and leadership to bring this program into existence has enabled me and other cohort members to push ourselves to exemplify leadership in educational technology. I am honored to have been one of her guinea pigs.

Finally, I also dedicate this work to my wife Gracie who not only encourages and enables me in my pursuits, but has also provided unquestioned support through sickness and health, through the highs and lows of professional and academic life, and shared the seemingly unending burden of my having yet another paper to write for my graduate studies.

My life has forever been changed, and made better, by the personal interventions of these four friends.

ABSTRACT

Morrow, M. Earnest, *Using knowledge elicitation techniques to establish a baseline of quantitative measures of computational thinking skill acquisition among university computer science students*. Doctor of Education (Instructional Systems Design and Technology), December, 2019, Sam Houston State University, Huntsville, Texas.

Purpose

The purpose of this study was to establish a baseline of quantitative measures of computational thinking skill acquisition as an aid in evaluating student outcomes for programming competency. Proxy measures for the desired skill levels were identified that reliably differentiate the conceptual representations of computer science students most likely, from those least likely, to have attained the desired level of programming skill. Insights about the development of computational thinking skills across the degree program were gained by analyzing variances between these proxy measures and the conceptual representations of cross-sections of participating students partitioned by levels of coursework attainment, programming experience, and academic performance. Going forward, similar measures can provide a basis for quantitative assessment of individual attainment of the desired learning outcome.

Methodology

The voluntary participants for this study were students enrolled in selected undergraduate computer science courses at the University. Their conceptual representations regarding programming concepts were elicited with a repeated, open card sort task and stimuli set as used for prior studies of computer science education. A total of 135 students participated, with 124 of these providing 296 card sorts. Differences between card sorts were quantified with the edit distance metric which provided a basis for statistical analysis. Card sorts from cross-sections of participants were compared and

contrasted using graph theory algorithms to calculate measures of average segment length of minimum spanning trees (orthogonality), to identify clusters of highly similar card sorts, and to reduce clusters down to individual exemplar card sorts. Variances in distance between the card sorts of cross-sections of participants and the identified exemplars were analyzed with one-way ANOVAs to evaluate differences in development of conceptual representations relative to coursework attainment and programming experience.

Findings

Collections of structurally similar card sorts were found to align with categorizations identified in earlier studies of computer science education. A logistic regression identified two exemplar sorts representing deep factor categorizations that reliably predicted those participants most, and least likely to have attained the desired level of programming skill. Measures of proximal distance between participants' card sorts and these two exemplars were found to decrease, indicating greater similarity, as students attained progressive coursework milestones. This finding suggests that proximal distances to exemplars of common categorizations for this stimuli set can effectively differentiate conceptual development levels of students between, as well as within, cross-sections selected by achievement of coursework milestones.

Measures of proximal distances to one exemplar of deep factor categorization were found to increase, indicating less similarity, as participants' levels of programming experience increased. This finding was contrary to the theoretical framework for skill acquisition. Further analysis found that variances in experience level as captured by the study instrument were not equally distributed among the cross-sections. The

preponderance of participants reporting greater levels of experience were degree majors not required to enroll in the courses most likely to develop that specific conceptualization. Therefore, for this deep factor categorization, instruction was found to have a greater influence on conceptual development than programming experience. However, it is possible that other categorizations, such as those related to software engineering technology, may be found to be more influenced by experience.

The orthogonality of participant card sorts was found to increase with each category of increase in academic performance, as in keeping with prior studies. Orthogonality also increased with greater levels of programming experience as expected by the theoretical framework. However, since experience was not equally distributed across categories of coursework achievement, the relationship between the orthogonality of participant card sorts and milestones of coursework achievement was not found to be statistically significant overall.

Conclusions

Based on the findings, the researcher concludes that a baseline of quantitative measures of computational thinking skills can be constructed based upon categorizations of elicited conceptual representations and associated exemplar card sorts. Eleven categorizations identified in a prior study of computer science seniors appear to represent reasonable expectations for deep factor categorizations. Follow up research is recommended (a) to identify for each categorization the exemplar card sorts that may be specific to different degree majors, and (b) to identify which categorizations may be more influenced by programming experience than by instruction.

Given an elicitation tool that prompts for the specific categorizations and a set of exemplar representations as proposed above, instructional programs can establish expected ranges of proximal distance measures to specific exemplars. These exemplars should be selected according to particular categorizations, degree majors, and coursework milestones. These differentiated measures will serve as evidence that students are meeting the instructional program learning objective for developing competency in the design and implementation of computer-based solutions.

KEY WORDS: Computer science education, Skill acquisition, Dreyfus model, Assessment, Programming, Computational thinking, Card sorts

TABLE OF CONTENTS

	Page
DEDICATION	iii
ABSTRACT	v
TABLE OF CONTENTS	ix
LIST OF TABLES	xiii
LIST OF FIGURES	xiv
CHAPTER	1
I INTRODUCTION	1
Background	1
Statement of the Problem	3
Theoretical Framework	5
Definition of Terms	12
Purpose of the Study	20
Significance of the Study	21
Research Questions	23
Null Hypotheses	24
Delimitations	24
Limitations	26
Assumptions	27
Organization of the Study	28
II REVIEW OF LITERATURE	29
Defining Programming Competencies	29

Assessing CT Competencies	31
Post-secondary Experiences with Programming Competencies	33
Computational Practices as Acquired Skills	36
Attaining Problem-solving Competency	37
Problem-solving Competencies in Computer Science	42
Summary	47
III METHODOLOGY	52
Research Questions	53
Null Hypotheses	54
Research Design	54
Selection of Participants	55
Instrumentation	58
Procedures	65
Data Analysis	69
Summary	78
IV RESULTS	81
Introduction	81
Research Questions	81
Null Hypotheses	82
Data Collection Results	83
Data Derivations and Transformations	84
RQ1 - Analysis of Relationship between Orthogonality and Categories of	
Achievement	91

RQ 2 - Identification of Card Sorts that Exemplify the Desired Level of	
Conceptual Development.....	94
RQ3 - Analysis of Variances between Proximity to Exemplar Sorts by	
Coursework Attainment.....	108
RQ 4 - Analysis of Variances Between Proximity to Exemplar Sorts by	
Programming Experience.....	114
Summary of Findings.....	119
V DISCUSSION AND CONCLUSION	122
Introduction.....	122
Discussion.....	126
Conclusion	141
REFERENCES	147
APPENDIX A.....	158
APPENDIX B	163
APPENDIX C	167
APPENDIX D.....	171
APPENDIX E	186
APPENDIX F.....	203
APPENDIX G.....	220
APPENDIX H.....	224
APPENDIX I	230
APPENDIX J	231
APPENDIX K.....	238

APPENDIX L	242
APPENDIX M	245
APPENDIX N.....	248
APPENDIX O	251
VITA.....	253

LIST OF TABLES

Table	Page
1 Computer Science Courses to be Solicited for Student Participation in the Study	57
2 Computer Science Cross-Section Details	58
3 Two Example Sorts.....	71
4 Transformation of Sort A to be Identical to Sort B	72
5 Pair-wise Edit Distances Among Four Sorts.	73
6 Logistic Regression Coefficients for Proximity to Exemplars 6 and 21	103
7 NMST by Performance Quartile as Reported in the McCauley et al. (2005) Study	128
8 Mean NMST Values for Introductory Students Versus Non-Introductory Students Categorized by GPA Ranking.....	131

LIST OF FIGURES

Figure	Page
1 Hypothesized framework for computational thinking skill acquisition.....	11
2 Stimuli set consisting of 26 single word programming terms for card sort use. ..	61
3 Initial screen display of card sort task.....	62
4 Card sort task underway.....	63
5 Example topological graph of a collection of four sorts with the edit distances between each pair of sorts displayed.	73
6 Illustration of minimum spanning tree.....	74
7 Greater differentiation of sorts represented by NMST measure.	75
8 Box plot of sort proximity to a probe.....	77
9 Bar chart of frequency counts from cross-tabulation of card sorts by coursework achievement category and category of orthogonality.	92
10 Illustration of multiple step process used to identify exemplar sorts of deep factor categorization for research question 2.	95
11 Participant characteristics for collection 13.....	96
12 Display of sort Id 4 from collection 13.....	98
13 Clique analysis results for collection 13 showing <i>d</i> values of 7 and 8.	100
14 Exemplar 6, representing the <i>Know / Don't Know</i> criterion.	105
15 Exemplar 201, from the <i>Concrete / Abstract</i> CAG.....	106
16 Exemplar 185, an intermediate <i>Types of Programming Terms</i> CAG example. .	107
17 Exemplar 21, an advanced <i>Types of Programming Terms</i> CAG example.	107
18 Surface factor sort grouping alphabetically (top) and by length (bottom).....	108

19	Box and whiskers plots of proximities to each exemplar sort by categories of coursework attainment.	110
20	Comparison of group means for proximity to exemplar sort 185.....	112
21	Comparison of group means for proximity to exemplar sort 21.....	113
22	Box and whiskers plots of proximities to each exemplar sort by categories of purposeful programming experience.	116
23	Comparison of group means for proximity to exemplar sort 185.....	117
24	Comparison of group means for proximity to exemplar sort 21.....	118
25	Bar chart of frequency counts from cross-tabulation of card sorts by coursework achievement category and category of orthogonality.	129
26	Bar chart of NMST means by category of GPA ranking (n = 68).....	130
27	Comparison of mean NMST of introductory and non-introductory participants categorized by programming experience (left) and by prior instruction (right).	132
28	Distribution (by percentage) of participants among the categories for programming experience and prior instruction between the introductory and non-introductory groups.....	132
29	Percentages of programming experience categories per coursework achievement category (n = 124).....	138
30	Percentage of experience level by intended majors.....	138
31	Percentage of intended majors for categories of coursework achievement.	139
32	Proximal distances to Exemplars 21 and 185 (all participants).	140
33	Proximal distances to Exemplars 21 and 185 (Introductory group).	140

34 Exemplar sort 21 for the <i>Types of Programming Terms</i> categorization.....	141
---	-----

CHAPTER I

Introduction

Computer programming is coming to be widely regarded as a desirable skill for current students and future workers. Coding is being referred to as the new literacy (Campbell, 2016), a notion that has been echoed by Bill Gates and Marc Prensky (Paul, 2016). It has also been claimed that by learning to code, students develop a cognitive skill set, known as computational thinking, which has benefits beyond programming (Wing, 2006). Consequently, a number of popular initiatives, such as the non-profit organizations Code.org, Girls Who Code, and Code2College, have emerged to promote the vision that computer science classes should be as available to all students as those for math and science (Code2College.org, 2017; Code.org, 2018a; GirlsWhoCode.com, n.d.). In recognition of this need, in the United States, the STEM Education Act of 2015 included computer science as an element of the science, technology, engineering, and mathematics curriculum for secondary education (Guzdial & Morrison, 2016).

Background

There are several specific motivations for this increasing demand for teaching and learning computer science. At the national level, educators, business leaders, and policy makers observe other nations placing an emphasis on teaching programming skills and suggest that the United States needs to do likewise to remain competitive in world markets (Campbell, 2016; Paul, 2016). Advocates for underrepresented populations seek to expand inclusion of these populations among those with programming skills (Adams & Reed, 2015; Google, n.d.). They view the field of computer science as a socioeconomic escalator providing opportunity for upward mobility (Campbell, 2016). At the individual

level, students and their parents, concerned with changing trends in the workforce, view computer-related jobs as more desirable and as offering better salaries than most other opportunities (Code.org, 2018b).

Among educators, the act of programming computers has a recognized pedagogic value. Papert (1980) viewed programming as a means for implementing constructionist theories of learning. The act of programming is one means of moving the learner from the passive role of consuming knowledge to an active role of creating and manipulating knowledge. The software environment is malleable to the needs of the learner and the learner develops an interactive relationship with the programming environment as the code is tested and debugged. Debugging is a constructionist process of repeatedly building, testing, and modifying an artifact and one's knowledge until specific objectives are met (Burleson, 2005).

Through influential essays by Jeanette Wing (2006, 2008) the pedagogic benefits ascribed to learning to program computers have become encapsulated as a learning objective for computational thinking. While a precise and widely accepted definition for computational thinking remains elusive (Denning, 2017; Selby, 2015), one common definition is that computational thinking is the set of mental processes used to formulate problems in such a way that they can be solved computationally (Czerkawski & Lyman, 2015). Computational solutions are defined as a specification of algorithms and procedural steps that can be implemented by a particular abstract model of computation (Aho, 2012). As a learning objective, computational thinking remains an active focus for research and discussion into appropriate methods of developing students' computational mental processes and of assessing their resulting computational solutions.

Statement of the Problem

At the post-secondary level, computational thinking is one desired learning outcome for computer science education. The Accreditation Board for Engineering and Technology (ABET) refers to this specific outcome, ABET Student Outcome 2, as the ability to develop computer-based solutions to meet specified requirements (ABET, 2019). That is, graduates of computer science degree programs should demonstrate competence in development of computational solutions and code implementations.

This desired learning outcome requires students to assimilate new conceptual knowledge while also acquiring complex cognitive skills (Guzdial & Morrison, 2016) for the transfer of this knowledge to the computational solution of authentic problems. Anecdotal and empirical evidence suggest that some portion of computer science students fail to attain this desired outcome (Casperson, Larsen, & Bennedsen, 2007; Dehnadi & Bornat, 2006; McCartney, Boustedt, Eckerdal, Sanders, & Zander, 2017) while a variety of causes for this failure continue to be debated (Ahadi & Lister, 2013; Patitsas, Berlin, Craig, & Easterbrook, 2016; Robins, 2010). Valid and reliable instruments for assessing student attainment of the desired level of competency in computational problem solving are required. However, multiple studies offer conflicting evaluations of frequently utilized methods (Lister, 2011; McCartney et al., 2013; McCracken, 2001). In a recent essay reviewing the current practices for assessing computational thinking skills, Denning (2017) summarizes the problem facing computer science educators: after a decade of research and academic discussion into computational thinking, “we have no consensus on what constitutes the skill and our current assessment methods are unreliable indicators” (p. 36).

Other fields of study at the post-secondary level have similar expectations and challenges regarding student competency with domain related problem-solving abilities. Research has been conducted into the acquisition of these skills in the domains of physics, chemistry, and biology (Bissonnette et al., 2017; Chi, Feltovich, & Glaser, 1981; Irby et al., 2016; Krieter, Julius, Bush, Scott, & Tanner, 2016). These studies have identified significant differences in the cognitive representations of domain knowledge between novices and those who have demonstrated competency in that field's required abilities (Bissonnette et al., 2017; Chi et al., 1981; Irby et al., 2016; Krieter et al., 2016). In the most recent of these studies, researchers have been able to hypothesize conceptual representations that they then validated as reliable indicators of expert-like problem-solving ability (Bissonnette et al., 2017; Krieter et al., 2016). The implication of these studies is that comparison of a student's conceptual representation of domain knowledge against these expert-like indicators can serve as a proxy measure of their degree of attainment of problem-solving competency.

Denning (2017) suggests computer science education follow a similar approach in his call to establish "guidelines for different skill levels of computational thinking to support" (p. 36) assessment of competency in this required skill. He proposes these guidelines be modeled after a framework for skill development, such as the Dreyfus five stage model of skill acquisition (S. E. Dreyfus & Dreyfus, 1980). Such a framework hypothesizes a series of identifiable changes in how a learner's knowledge is organized as the learner gains experience in the skill through repeated practice over time (S. E. Dreyfus & Dreyfus, 1980; Gladwell, 2008; Polanyi, 1966). As with the chemistry and biology studies, by comparing the conceptual representations of computer science

students against this series of expected changes an improved method for assessing computational skill development can be realized which is based upon a progression of indicators associated with sequential levels of skill acquisition.

The prior studies of domain related problem-solving skills in physics, biology, and chemistry were able to hypothesize expert-like conceptual representations for problem solving based upon well-established foundational principles in those domains. For the field of computer science, an alternative approach may be more practical while equally effective. Conceptual representations can be elicited from cross-sections of computer science students and analyzed to identify those that differentiate the higher performing seniors (McCauley et al, 2005) from other samples of the learner population. These identified conceptual representations can designate attainment of the desired ABET student outcome 2. Subsequently, comparison of learners' conceptual representations against these indicators can serve as a proxy measure of their degree of attainment of computational problem-solving competency.

Theoretical Framework

Personal Construct Theory and knowledge elicitation. Personal construct theory addresses how individuals create cognitive representations of their environment based upon their experiences, and then use these representations to recognize and respond to their current situation (Kelly, 1955). These cognitive representations of events, or constructs, denote an individual's categorization of stimuli (Bell, 2011). As individuals each have unique experiences in the world, their categorizations of the world will be unique to them (Upchurch, Rugg, & Kitchenham, 2001). For a set of events shared by a group, such as a learning activity, individual categorizations should be similar enough to

enable meaningful communication and shared understanding, yet different enough to reflect personal interpretation and expression (Upchurch et al., 2001).

By eliciting the personal constructs shared by members of a group and examining the similarities and differences among the constructs, it is possible to gain an understanding of the event and the meaning attributed to it by each individual (Fincher & Tenenberg, 2005; Rugg & McGeorge, 2005). Several techniques, such as repertory grid, laddering, and card sorting, described by Kelly in his description of personal construct theory (1955), are capable of eliciting knowledge than the more common qualitative methods of interviewing and surveying are able to collect (Upchurch et al., 2001). Furthermore, the repertory grid and card sorting approaches collect data suitable for statistical analysis (Deibel, Anderson, & Anderson, 2005; Rugg & McGeorge, 2005), thus enabling quantitative methods of analysis to gain qualitative-like understanding of respondents' construction of knowledge and meaning.

Card sorting is a categorization task (Fincher & Tenenberg, 2005) where a participant sorts elements of a stimuli set into multiple groups based upon a sorting criteria (Rugg & McGeorge, 2005). This active construction by the participant of external groups, i.e. categories, reflects their internal conceptualizations (Fincher & Tenenberg, 2005). Card sorts are especially well suited for capturing data of non-scalar or nominal categories, for which repertory grids are not suitable (Rugg & McGeorge, 2005). In an open, or unframed, type card sort, participants are allowed to choose the sort criteria, and to specify category labels. Thus, as a data collection method, open card sorts have the benefit of being participant-centric rather than researcher-centric (Fincher & Tenenberg, 2005). Administratively, card sorts have an advantage of being easy to use, relatively

quick, and systematic (Rugg et al. 1992). For analysis, card sort results reveal participants' categorizations which can be compared to identify similarities and differentiations in participant conceptualizations (Rugg & McGeorge, 2005).

Card sort studies have been used to examine the differences in conceptualizations between novices and experts (Chi et al., 1981). Results support that experts have a larger and more elaborate conceptualization schema of their field of knowledge (Rugg & McGeorge, 2005). This is in keeping with personal construct theory in that an individual's cognitive representations of experiences allows for current externally presented events to be categorized according to previously seen criteria, identified in context, and used as a basis to select an appropriate response (Chi et al., 1981; Kelly, 1955). An expert's schema provides a more differentiated filter for recognition of fragmentary cues found in a problem presentation. This greater ability to differentiate among personal constructs during categorization of unstructured stimuli may largely explain the expert's greater ability for problem-solving (Chi et al., 1981).

Dreyfus model of skill acquisition. The Dreyfus model of skill acquisition describes changes in cognitive processing as a learner progresses in experience and ability from novice to expert (H. L. Dreyfus, Dreyfus, & Zadeh, 1987; S. E. Dreyfus & Dreyfus, 1980). The model was originally developed to guide instructional design of training programs for high-order skills such as aircrew emergency decision making (S. E. Dreyfus & Dreyfus, 1980). Based on phenomenological research and philosophical deliberation (Peña, 2010) the model argues that instructional strategies must be adapted to a learner's current skill level stage in order to facilitate progression to the next stage (S. E. Dreyfus & Dreyfus, 1980). The model also emphasizes the role that long-term,

repeated experience plays in skill progression. As such, the model reinforces the adage that skill is acquired over time with practice (Denning, 2017; Polanyi, 1966).

The model describes skill acquisition as successive transformations of mental functions. The transformations are sequentially dependent so that they must occur in a fixed order (S. E. Dreyfus & Dreyfus, 1980). The stages of the resulting model are described as follows.

Novice. Initially a learner has limited knowledge of and no experience with the skill. This is the Novice stage. Performing the skill is a matter of using explicit knowledge to identify certain external stimuli, and to match the stimuli to a prescribed set of unique conditions that then dictate the action to be performed. Thus, performance is strictly rule-based. The learner's awareness is focused on monitoring performance for conformity and compliance. In the Novice stage, each of the mental processes is in a primitive form which requires conscious thought on the part of the learner (S. E. Dreyfus & Dreyfus, 1980). This creates a high cognitive load for the learner. Performance of the skill is methodical (Carraccio, Benson, Nixon, & Derstine, 2008). An example is initially learning to drive a car: conditions inside and outside of the car must be continuously monitored; sensory feedback processed and corrective actions taken; and rules followed. This is also the common state for introductory computer science students.

Advanced Beginner. As the Novice practices the skill, the more frequent rules and actions develop automaticity. This lessens the cognitive load of performance on the learner. At this stage the learner is an Advanced Beginner (H. L. Dreyfus et al., 1987) and can begin to filter stimuli and prioritize responses based on relevance (Carraccio et al., 2008). In the example of learning to drive a car, many of the driving procedures such as

preparing to drive, accelerating and braking the car, and monitoring the environment inside and outside of the car become routine and require a minimum of conscious thought.

Competent. In the Novice and Advanced Beginner stages, the conditions of stimuli are free of context. With more experience, the learner begins to recognize recurring meaningful patterns among stimuli conditions. These patterns represent a growing awareness of situational contexts which are derived from recognition of prior examples. An appropriate and more complex response is directly triggered by recognition of the particular situation. At this stage the learner has achieved Competence (S. E. Dreyfus & Dreyfus, 1980). The skill performance is now situated within the context of a larger picture (Carraccio et al., 2008). In the example of learning to drive a car, the driver is now aware of the routine traffic flow surrounding the car and how the flow varies according to location and time of the day.

Proficient. Continued experience and practice exposes the competent performer to a widening variety of typical situations. The scope of the learner's awareness of situations grows to encompass whole situations which conclude with the achievement of a goal. The learner begins to assess situations according to their relevance to desired objectives. Thus, a single recognized situation, encountered at different points in time during pursuit of different objectives will be treated as different situations (S. E. Dreyfus & Dreyfus, 1980). This is the Proficiency stage of skill development. The Proficient performer identifies multiple possible solutions to a situation and evaluates the options from the perspective of a particular goal. As an example, a proficient driver is able to evaluate alternative routes to a desired destination based upon the expected traffic flows

of locations along the way at the particular time of day and under pertinent weather conditions.

Expert. Eventually, the performer may gain such a vast repertoire of contextually experienced and perspective-specific situations that the selection of complex courses of action develops automaticity to the extent of appearing to be intuitive (S. E. Dreyfus & Dreyfus, 1980). This is the stage of Expertise in the skill. Experts perceive features that are atypical from the norm and will notice the unexpected (Carraccio et al., 2008).

Hypothesized framework for Computational Thinking. Individually, the Dreyfus model of skill acquisition and Kelly's personal construct theory each account for the significant differences documented between the endpoints of Novice and Expert (Chi et al., 1981). Combined together, they can form a framework that suggests specific ways and steps in which the conceptualizations of the Novice transform into those of an Expert. Such a description of the intervening levels of skill acquisition can directly address Denning's (2017) call for guidelines to the different skill levels of computational thinking. Such a combined hypothetical framework for the acquisition of computational thinking skill is presented in Figure 1. As computer science students practice their programming and gain experience, their cognitive representations of the programming domain should grow more sophisticated. Figure 1 denotes this growth in both the knowledge content that is retained, and that which is transferrable, relative to the five stages of skill acquisition. Each of these stages is described more fully below.

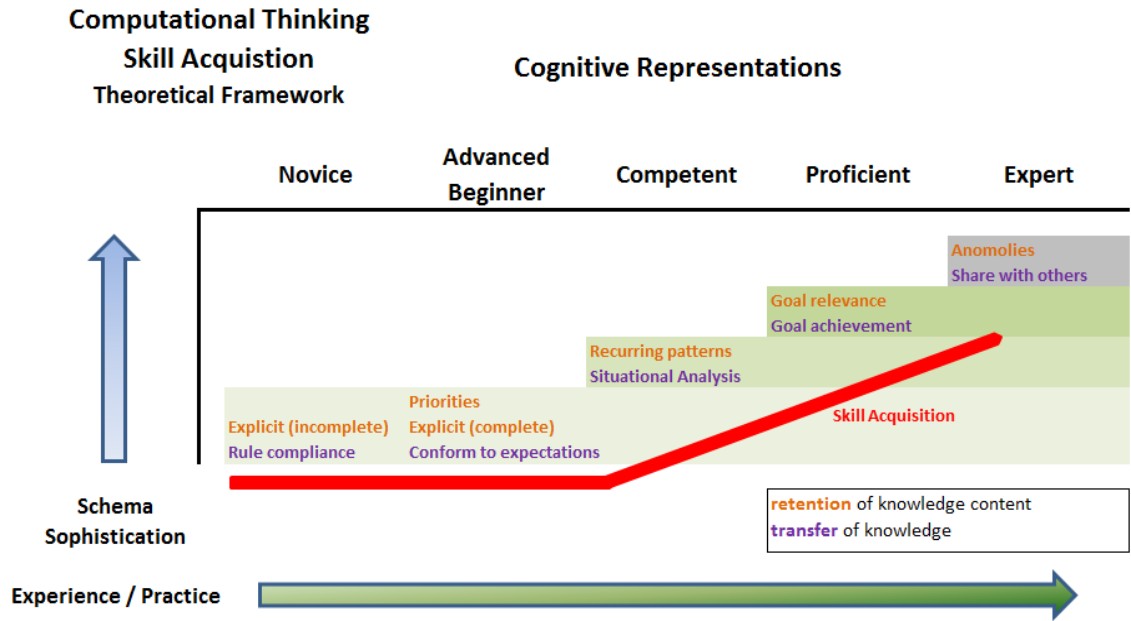


Figure 1. Hypothesized framework for computational thinking skill acquisition.

The Novice is taught a pre-defined schema of stimuli related through a two-dimensional causal network (Carraccio et al., 2008). The constructs are context-free and there are few if any abstractions. The constructs and causal network are under active construction as they are initially learned and reinforced through practice. Therefore, categorizations will be largely based on superficial aspects of the prescribed stimuli and may be incomplete, i.e., the novice may not be able to sort all elements of a stimuli set into categories. Cognitive processes are engaged in monitoring rule compliance.

The Advanced Beginner has completed the construction of the prescribed schema. The causal network can now be methodically referenced and followed with minimal cognitive load which allows for cognitive processes for prioritization and conformance. Categorizations remain based on superficial aspects of the constructs but are complete. The advanced beginner computer science student is able to consider tradeoffs in execution time or code clarity when considering language syntax options.

Competence is achieved as patterns of related stimuli conditions are recognized as distinct recurring situations, and responses become differentiated according to the situation. The constructs of the beginner's cognitive schema become supplemented with abstractions representing these situational patterns. Increasingly complex interrelationships among the constructs enable greater differentiation which can be directed toward situational analysis. Categorizations become less superficial and begin to reflect contextual situations. The competent computer science student is able to associate desired functions such as search, sort, or tree traversal to typical code fragments.

Proficiency is marked by the addition of constructs representing relevance to end-goals. This enables differentiation between alternative responses based upon criteria for goal achievement. Categorizations become more numerous, more differentiated, and more abstract. The proficient computer scientist is able to integrate multiple functions and the associated code fragments into a processing system that meets project-level objectives.

The schema of the Expert is very robust in terms of constructs and relationships. Abstractions exist for stimulus conditions whose presence or absence has proven to be significant in the past. Processes for transfer of knowledge reach beyond individual needs to benefit others. Expert categorizations will be the most numerous and most differentiated.

Definition of Terms

Accreditation Board for Engineering and Technology (ABET). In the United States, ABET is the accrediting organization for post-secondary education programs in engineering and technology, including computer science. ABET establishes the criteria

and the process for accreditation of an institution's programs (ABET, 2019). Specific to this study, ABET criteria define student outcomes for computer science education and the requirement for programs to self-evaluate achievement of the objectives as part of an ongoing continuous improvement process.

Baseline indicators of skill acquisition progress. This study proposed to associate the desired level of skill acquisition for post-secondary computer science students with specific structural exemplar sorts of the stimuli set used for this study. The stimuli set was a list of 26 single word programming terms, such as constant, variable, loop, and list, that have been used previously (McCauley et al., 2005; Sanders et al., 2005) with computer science students as knowledge elicitation prompts. This study hypothesized that as students' progress in their skill development then their categorizations of the stimuli set will become more differentiated and be evidenced as changes in the structural exemplar sorts as derived from cluster analyses of their card sorts. These structural exemplar sorts can then be used as a baseline of skill acquisition indicators to gauge students' progress towards attainment of the desired level of expertise. Thus, if a student produces card sorts that are highly similar to the structural exemplar sorts of students who have been deemed to be at the desired level of skill acquisition, then that student's card sort result will be taken as an indicator of having attained the ABET student outcome 2.

Bloom's taxonomy. Specific to this study, Bloom's revised taxonomy of the cognitive domain specifies a hierarchy of cognitive tasks arranged by cognitive demand (Selby, 2015). The Bloom levels of cognitive tasks ordered from least demanding to the most demanding are: remembering; understanding; applying; analyzing; evaluating; and

creating (Anderson et al., 2001). Assessments of retention of knowledge require cognitive tasks at the levels of remembering and understanding. The remaining levels involve cognitive tasks related to the transfer of knowledge (Anderson et al., 2001).

Categorization. Categorization is an elemental cognitive process for collecting similar objects together. Specific to personal construct theory (Kelly, 1955) individuals make sense of the world by categorizing the objects sensed within it. An individual will group objects together in a category based upon a perceived similarity, known as the criterion. This criterion also differentiates objects in this group from objects in other groups (Rugg & McGeorge, 2005).

Cluster analysis. A cluster analysis of card sorts seeks to discover similarity patterns in a collection of sorts. A cluster is a subset of the sort collection where the sorts within the cluster share a similarity. Qualitative methods of cluster analysis are based on lexical analysis of the category names and are represented with dendograms. Quantitative methods of cluster analysis of card sorts are based on structural similarity as measured by the edit distance (Precht, Szwillus, & Domik, 2014). Data mining techniques have developed several algorithms for quantitative identification of clusters: such as, *d*-cliques; *k*-means, hierarchical clustering, and multidimensional scaling (Deibel et al., 2005; Precht et al., 2014). This study utilized an algorithm for *d*-cliques.

Cognitive conceptualizations and representations. For the purpose of this study these terms refer to the cognitive framework an individual constructs to organize and utilize knowledge as described in Piaget's theory of cognitive development (Piaget & Inhelder, 1969).

Cognitive processes and tasks. Cognitive processes and tasks are an individual's mental procedures for utilizing the schema of knowledge. One enumeration and classification of these processes can be found in Blooms' revised taxonomy of the cognitive domain (Anderson et al., 2001). The majority of these processes involve transfer of the knowledge to new requirements, problems, and situations.

Conceptual expertise. From prior studies of this type, conceptual expertise is the level of skill acquisition and cognitive processing of putative experts in the field (Krieter et al., 2016). One learning objective for computer science students at the post-secondary level is to demonstrate an advanced level of ability for program design and implementation, similar to that of professionals. For the purpose of this study, conceptual expertise refers to the progression of skill acquisition beyond the novice and advanced beginner levels as described by the theoretical framework.

Cumulative GPA in Computer Science (CS) coursework. This measure is the grade point average of all of the computer science courses a student has completed to date at the institution.

Edit distance. The edit distance metric is an indication of the structural similarity between exactly two sorts. The metric is calculated as the fewest number of elements of the stimuli set that must be moved from one category to another in order to make the sorts identical (Deibel et al., 2005). An edit distance of zero indicates the sorts are identical in that the stimuli set has been organized into the same set of groups, even though the groups may have been given different category names by the participants. The larger the edit distance value becomes, the greater the dissimilarity between the sorts.

Geek gene. The hypothesis of the geek gene is that there are two types of computer science students: those who can learn to code; and those who can't. The differentiator between the two groups being an innate ability apparently due to the presence or absence of the geek gene (McCartney et al., 2017). This hypothesis is also associated with the perception that the distribution of grades in an introductory CS course is bi-modal, with students either performing very well, or very poorly. Recent research refutes both of these perceptions (Ahadi & Lister, 2013; Patitsas et al., 2016; Robins, 2010).

Graduating CS seniors. For the purpose of this study, graduating seniors are computer science students who have completed 28 hours or more of computer science coursework with a cumulative GPA of at least 2.0 out of 4.0.

Introductory CS course. The introductory computer science course at the participating institution is commonly referred to as Java 1. Learning objectives for this course are to learn the fundamental elements of the Java programming language, learn the process for developing a functionally correct program, and apply the concepts and procedures of object-oriented programming to the creation of solutions to realistic problems and needs (Sam Houston State University, 2018). Current enrollment in this course will be categorized as Course A in the milestones of coursework achievement.

Intermediate milestones of coursework attainment. Prior studies have identified differences in the conceptual expertise of seniors versus students of introductory courses. This study proposed to investigate how these differences manifest as students progress through milestones of coursework achievement, such as the end of each school year. However, at the participating institution, the different computer science

degrees have unique schedules of courses (tracks) throughout their programs.

Additionally, as some students will change majors, or may need to alter their individual class schedules, not all students will have had the same set of courses at the end of each semester or school year. Therefore, milestone progress through the degree programs will be determined based upon current enrollment in, or completion of select courses that are sequentially dependent on one another for each particular degree program. For this study, three intermediate points that follow the introductory course A will be determined for each degree program, and be categorized as courses B, C, and D.

Knowledge elicitation. Knowledge elicitation is data collection from an individual of the cognitive representation of knowledge which they uniquely possess. The data collection tools for knowledge elicitation include repertory grids, laddering, and card sorting. For the purpose of this study knowledge elicitation of computer science students will be accomplished with use of an open card sorting task.

Orthogonality of card sorts. Orthogonality is a measure of the degree of differentiation among sorts in a collection. The measure is based on the edit distance between every pair of sorts in the collection. Orthogonality is a proxy measure for an individual's conceptual expertise. Sorts of Novices tend to exhibit low orthogonality while the sorts of experts tend to exhibit high orthogonality.

Personal constructs. Based on Kelly's (1955) personal construct theory, people make sense of the world by categorizing it. Constructs are the attributes an individual associates with objects in order to describe and categorize them.

Programming experience. As computer science students progress through their degree program they add to their experience in developing, testing, and implementing

computer programs. Some of this time may occur prior to their entering the program, some occurs as programming assignments for course homework or lab periods, and some may be for personal or professional purposes outside of their formal education. The amount of time varies for each individual. For this study, the students responded to several survey questions regarding prior learning in programming and to Likert-scale assessments of coding experience they gained with various types of projects and assignments. These responses were evaluated by the researcher for their indications of overall experience and will result in an experience categorization for the student of Light, Moderate, or Extensive.

Quantitative vs qualitative measures of conceptual expertise. A quantitative approach to assessing conceptual expertise in a field of knowledge is based on the edit distance metric to analyze the similarity or dissimilarity among two or more card sorts. Analyses that can be derived with the use of the edit distance are the orthogonality of a collection of sorts; clusters of sorts that are within a specified distance of each other, and the proximity of a specific sort to a target or exemplar sort. These comparisons consider only which stimuli are grouped together, and are ignorant of any category names ascribed to the groups by the participants. Hence, these measures are comparisons of the similarity of the sort structures. They can be objectively determined and computer generated, so they scale well with large sample sizes.

A qualitative approach to assessing conceptual expertise in a field of knowledge is based on evaluating a learner's demonstration of transferring the learning by performing the skill. For computer science students these demonstrations take the form of creating a segment of computer code that performs a specified function. These skillful actions are

often assessed with an assignment specific rubric identifying the criteria and scale for evaluating proficiency of the performance. However, these instruments require human judgements and therefore do not scale well to large samples (McCracken et al., 2001). This study sought to establish the quantitative approach as a viable alternative to the qualitative approach for assessment of conceptual expertise.

Proximity of card sorts. The proximity of one sort to another refers to the structural similarity of the two sorts. This measure is expressed as the edit distance between the two sorts. Specific to this study, the proximity measure of individuals' sorts to an exemplar structural sort may be taken as an indication of which individuals have achieved a skill baseline and which have not.

Schema. For the purpose of this study schema refers to the cognitive framework an individual constructs to organize and utilize knowledge as described in Piaget's theory of cognitive development (Piaget & Inhelder, 1969).

Skill acquisition. Skills in the cognitive domain involve tasks associated with transfer of knowledge such as applying, analyzing, evaluating, and creating (Anderson et al., 2001). Individual skills improve as they are repeated or practiced and there is a progression to this improvement. Thus, skills are not just learned, but are acquired and improved over time through experience (Denning, 2017; Polanyi, 1966). According to the Dreyfus model, skill acquisition progresses through five stages beginning with Novice and ending at Expert. Students enrolled in higher education computer science degree programs are assumed to begin at the Novice stage. However, there is no research literature regarding their typical stage upon graduation. Furthermore, it can be assumed to be unlikely that they graduate as Experts. Thus, for this study, student progression

through skill development was analyzed in terms of milestones of coursework attainment, and levels of programming experience.

Stimuli set. For card sorting knowledge elicitation, each participant is presented with a set of cards, where each card contains an entity. These entities may be pictures, single words or phrases, or descriptions of individual situations or problems. Entities provide the participant with a stimulus to cognitively categorize that entity, so the set of all the entities to be categorized is referred to as the stimuli set. Specific to this study, the stimuli set was 26 single-word programming terms utilized in previous studies (McCauley et al., 2005; Sanders et al., 2005) of conceptual expertise in computer science students. Example stimuli include terms like constant, variable, loop, and list.

Structural exemplar sort. Sorts within a cluster share a structural similarity as they are all within a specific edit distance of each other. The single sort in the cluster with the smallest total edit distance to all the other sorts in the cluster (i.e., closest to being equidistant) is the structural exemplar (McCauley et al., 2005). The structural exemplar sort is to the cluster as a mean value is to a sample of measurements.

Purpose of the Study

The purpose of this study was to establish a baseline of quantitative measures of computational thinking skill acquisition as an aid in evaluating student outcomes for programming competency. Proxy measures for the desired skill levels were identified that reliably differentiate the conceptual representations of computer science students most likely, from those least likely, to have attained the desired level of programming skill. Insights about the development of computational thinking skills across the degree program were gained by analyzing variances between these proxy measures and the

conceptual representations of cross-sections of participating students partitioned by levels of coursework attainment, programming experience, and academic performance. Going forward, similar measures can provide a basis for quantitative assessment of individual attainment of the desired learning outcome.

The study used the knowledge elicitation instrument (a repeated, open card sort of 26 programming terms) and quantitative methods from McCauley et al. (2005) and Fossom and Haller (2005). The theoretical framework hypothesized that learners' organization of knowledge changes as learners gain experience with a skill over time (S. E. Dreyfus & Dreyfus, 1980), and that these changes are reflected in the elicited conceptual representations (Kelly, 1955). Therefore, cross-sections of students in the computer science degree program at the participating institution were selected for elicitation of conceptual representations reflective of their attainment of specific milestones of coursework, level of programming experience, and category of academic performance. A cross-sectional analysis approach similar to that used by Krieter et al. (2016) and Bissonnette et al. (2017) compared differences between groups to identify differentiating indicators of progression in computational thinking skill acquisition across the degree program.

Significance of the Study

Most immediately, a baseline of indicators of cognitive development of computational thinking resulting from this study can serve as an additional basis for formative assessments of student progress toward the ABET student outcome 2 for an ability to develop computer-based solutions to meet specified needs (ABET, 2019). As compared to the common practice of qualitatively assessing programming assignments,

the comparison of knowledge elicitation data against the baseline indicators will yield objective, quantitative, computer-assessed measures of progress toward skill acquisition. Such measures will enable timely identification of those students not progressing as expected and will provide insights for the design of appropriate intervention measures. This outcome will be of benefit to the students and potentially to the prospective employers of the students as they may have greater assurance of students' competency as programmers and analytical problem solvers.

Computer science education programs are required to demonstrate that program effectiveness is being evaluated, documented, and used as a basis for program improvement (ABET, 2019). A baseline of cognitive development indicators for formative assessments will provide additional means and measures for institutions to address this ABET accreditation criteria. This outcome will be of benefit to the institution.

Perhaps most importantly, this study, as a direct response to Denning's suggestion (2017) to assess computational thinking as a skill, addressed a gap in the research and discussion of computational thinking. While the studies of conceptual expertise in the fields of physics (Chi et al., 1981), biology (Bissonnette et al., 2017), chemistry (Krieter et al., 2016), and computer science (Fossum & Haller, 2005; McCauley et al., 2005; Sanders et al., 2005) have confirmed the ability of knowledge elicitation instruments to identify differences in conceptualizations between novices and experts, these studies have not, however, investigated how these changes develop and manifest during the continuum of learning and experience building that takes place between these endpoints. Insights gained into this development process will inform adaptive instructional strategies, as was

an original intention of the Dreyfus paper (1980). Furthermore, these insights may also direct future research into the nature of the computer programming activity, and the role that innate ability or some form of multiple intelligence factor (McCartney et al., 2017) contributes to accomplishment of these tasks. Better understanding of these factors will allow a more objective assessment and discussion of the benefits claimed for computational thinking.

Research Questions

The following research questions were addressed in this study:

1. Is there a relationship between categories of coursework achievement (Introductory, Mid-Program, Completing) and categories (High, Low, or Zero) of card sort orthogonality?
2. How many reliable and interpretable structural exemplar sorts are derivable from the collections of sorts produced by those participants most likely to have attained the desired level of programming skill?
3. Is there a statistically significant difference among the categories of computer science students' progression through milestones of coursework attainment (Introductory, Completing, and Mid-program) on the dependent variable, the edit distance between the card sorts to the exemplar sorts?
4. Is there a statistically significant difference among the categories of computer science students' programming experience (Light, Moderate, Extensive) on the dependent variable, the edit distance between the card sorts to the exemplar sorts?

Null Hypotheses

- H₀₁:* There is no statistically significant relationship between categories of coursework achievement (Introductory, Mid-Program, Completing) and categories (High, Low, or Zero) of card sort orthogonality.
- H₀₂:* No reliable and interpretable structural exemplar sorts are derivable from the collections of sorts produced by those participants most likely to have attained the desired level of programming skill.
- H₀₃:* There are no statistically significant differences among the categories of computer science students' progression through milestones of coursework attainment (Introductory, Completing, and Mid-program) on the dependent variable, the edit distance between the card sorts to the exemplar sorts.
- H₀₄:* There are no statistically significant differences among the categories of computer science students' programming experience (Light, Moderate, Extensive) on the dependent variable, the edit distance between the card sorts to the exemplar sorts.

Delimitations

Delimitations of a study describe the choices made during the design of the study by the researcher to establish practical boundaries. These decisions are both inclusionary and exclusionary in nature. They are presented to provide full disclosure of options that were considered, and those that were taken.

1. While the focus of the study is on assessment of student ability as described in ABET student outcome 2, the study also implicitly equates this outcome to computational thinking. Computational thinking is generally considered to

encompass additional cognitive processes. However, this study justifies its equivalence of the two terms on the basis that computational thinking processes exemplify a transfer of the ability to design and develop computer-based solutions to meet specified needs. Hence, the ABET student outcome 2 is essential to computational thinking.

2. While computational thinking is introduced via the K-12 educational reform movement, the study focus is on post-secondary computer science education only. This reduction in scope was elected in order to avoid issues of informed consent with underage populations.
3. Participants will be chosen from a single institution of higher education. The researcher had direct access to both the students and faculty of this University. As a practical matter, this access simplified the Institutional Review processes and expedited data collection. However, this decision could lessen transfer of the findings to other situations.
4. Participants self-reported their cumulative programming practice time and these data will not be verified against other sources.
5. The research questions were chosen with consideration for quantitative analysis with the collected data and in order to connect prior research with the ABET student outcome 2 and a theoretical framework that emphasizes the importance of experience in skill acquisition.
6. The card sort elicitation instrument is the same one used in the 2005 card sort studies of computer science conceptual expertise (Fossum & Haller, 2005; McCauley et al., 2005; Sanders et al., 2005). It was decided to use the identical

stimuli set and to attempt to replicate the results of these studies in order to minimize concerns regarding the validity and reliability of the instruments and procedures of this study. Future research may alter the stimuli set in order to pursue more focused research objectives.

7. A longitudinal study design may be more appropriate for researching the progression of knowledge and skill acquisition. However, such a design would require a much longer timeframe for data collection. Such a timeframe was outside the boundaries of the dissertation that encompasses this study.
8. This study did not correlate its quantitative assessments of students' achievement of the desired ABET student outcome 2 with other institutional evaluations of that outcome. While such a correlation is to be desired, it was decided to leave that as a worthy question for subsequent research.
9. Due to the size of the population, the card sort activity was administered online without direct supervision or intervention by the researcher. This could have permitted some invalid or incomplete responses by participants to be improperly included for analysis.

Limitations

Limitations describe potential weaknesses in the research, many of which are outside the control of the researcher. Some weaknesses are always inherent in the methods and procedures that are selected, while others become evident during the course of conducting the study. This statement of limitations discloses those conditions which might have affected the findings or which may be of interest for exploration in future research:

1. Knowledge elicitation techniques, such as card sorts, are proxies and not direct methods for assessing an individual's internal conceptualizations regarding a body of knowledge. The accuracy of these proxies is assumed based upon theory, observations, and prior use in research.
2. The measure of participants' cumulative programming practice or experience is only an approximation. As these data were self-reported, the variance in each approximation differed among participants. Therefore, findings based on cumulative practice time were affected to an unknown extent by this limitation.
3. Prior studies of differences in cognitive expertise among post-secondary students have compared only the endpoints on traditional four-year degree programs. The assumption has been made in these prior studies that these endpoints represent novice and expert levels. While this study aspired to demark this progression at intermediate milestones of the program coursework, it was not possible to calibrate these findings to a continuum from novice to competent or proficient programmer according to the Dreyfus model.

Assumptions

This study relies on the following assumptions:

1. That "individuals construct meaningful internal categories to reflect their understanding of distinctions in the world" (Fincher & Tenenber, 2005, p. 90).
2. Novices and experts categorize field related knowledge differently (Chi et al., 1981; Rugg & McGeorge, 2005).
3. "People learn a skill only by engaging with it and practicing it" (Denning, 2017, p. 36).

4. Cumulative grade point average in computer science coursework reliably reflects both a student's conceptual knowledge and computational thinking ability.

Organization of the Study

This study of using knowledge elicitation techniques to establish a baseline of quantitative measures of computational thinking skill acquisition among university computer science students is organized into five chapters. Chapter 1, the introduction, includes the study background, statement of the problem, the theoretical framework as derived from personal construct theory and the Dreyfus model of skill acquisition, purpose of the study, significance of the study, research questions, hypotheses, definition of terms, delimitations, limitations, assumptions, and organization of the study. Chapter 2 provides a review of the literature. Chapter 3 describes the methodology used for the study which includes an introduction, the research questions, hypotheses, research design, participants, data collection, data analysis, and a summary of the study proposal. Chapter 4 presents the results of the study including an introduction, the research questions, hypotheses, the overall descriptive statistics for the collected data, and the statistical results for each of the four research questions. Chapter 5 interprets the implications and meaning of the results as they relate to existing literature, to the theoretical framework, to the implications for teaching, learning, and assessing computer programming and computation thinking skills and abilities, and makes recommendations for future research.

CHAPTER II

Review of Literature

The study area for this dissertation is Computer Science education. Specifically, it focuses on an objective for developing students' competency in programming at the post-secondary level according to accreditation criterion for student outcomes. Learning outcome 2 of the Accreditation Board for Engineering and Technology (ABET, 2019) states:

Graduates of the program will have an ability to . . . design, implement, and evaluate a computer-based solution to meet a given set of computing requirements in the context of the program's discipline. (Criterion 3. Student outcomes).

This chapter summarizes the literature regarding academic debate and research into the nature of programming competency, and methods of assessing learners' attainment of the objective.

Defining Programming Competencies

The ability to write computer programs that accomplish beneficial tasks is frequently linked to the development of analytical and critical thinking skills. Papert (1980) asserted this relationship in writing about children working with the LOGO programming language. He coined the term "computational thinking" to describe the collection of cognitive activities of logical thinking, problem solving, and creativity which he believed were engendered by learning to program.

Jeannette Wing popularized the computational thinking (CT) term with an influential essay (2006) which advocated for the inclusion of CT in secondary and even primary education. She declared CT to be an essential learning objective for all students.

Supporting her reasoning is the view that programming which meets an authentic need is a problem-solving task and as such, is a model of constructionist learning activity.

Furthermore, Wing asserted that computational thinking involves multiple levels of abstraction and is, therefore, universally applicable as a method by which humans solve problems (Wing, 2006).

Initial efforts to describe computational thinking competencies as learning objectives were undertaken by educational standards organizations such as the Computer Science Teachers Association, the British Computer Society, and the International Society for Technology in Education (Denning, 2017). These included imprecise terms such as abstraction, decomposition, algorithms, and analysis to describe the cognitive process expected to be evident in the learning outcomes. Academic discussions of the time struggled for consensus on operationalized terminology for computational thinking. However, until CT could be clearly defined, corresponding pedagogical methods and assessment means could not be agreed upon (Zhong, Wang, Chen, & Li, 2016).

By 2012, consensus began to coalesce around a multi-part definition for computational thinking. A theoretical framework was proposed which operationalized CT in three dimensions: computational concepts, computational practices, and computational perspectives (Brennan & Resnick, 2012). This work coincided with an essay on computation and computational thinking by Aho (2012) that narrowed the definition of CT to the set of thought processes used to formulate a problem such that it can be solved computationally (Czerkawski & Lyman, 2015). Furthermore, these computational solutions are specifications of algorithms and procedural steps which can be implemented by a particular computational model (Aho, 2012). A computational model is an abstract

mechanism with prescribed capabilities, capacities, and procedures. These attributes of the computational model correspond to the set of computational concepts identified in Brennan and Resnick's framework (2012), such as sequences, loops, parallelism, events, conditionals, operators, and data. The set of thought processes involved in formulating a problem for computational solution correspond to the computational practices of the framework, such as incremental and iterative building, ongoing testing and debugging, improving through abstracting and modularizing, and reusing and remixing code (Brennan & Resnick, 2012). Thus, learning objectives for computational thinking may be operationalized as separate and distinct strategies for promoting knowledge retention of the computational concepts, and for utilizing the computational practices for knowledge transfer in authentic situations.

The computational concepts enumerated by the Brennan and Resnick framework may be considered fundamental knowledge constructs for most programming languages. The methods to promote and assess the retention of these concepts in students are well-established. However, it is the methods for promoting and assessing computational practices that have been the most debated and studied.

Assessing CT Competencies

In proposing their framework, Brennan and Resnick intended it to provide a basis for researching and evaluating development of computational thinking in learners (2012). They sought methods which could assess against two or more dimensions simultaneously. Their article elaborated on experiences with three techniques (a) a portfolio analysis, (b) an artifact-based interview, and (c) the design scenario. They recommended the design scenario method as most suitable for assessing both

computational concepts and computational practices without being too labor-intensive (Brennan & Resnick, 2012).

The design scenario is a problem-based interview where a participant is presented with code for a small application and asked to explain what the application does, to suggest ways to extend it, to identify and correct an embedded bug, and to add a new feature (Brennan & Resnick, 2012). Variations in the structure of design scenario tasks were subsequently studied empirically for the effectiveness and efficiency of administration (Zhong et al., 2016). These variations were defined in the two dimensions of directionality and openness. Forward direction tasks ask participants to create a coded solution from stated requirements. Alternatively, reverse direction tasks ask the participant to debug a code segment. Openness of a task refers to whether there are one or more correct outcomes and expected processes for arriving at the outcome. Closed tasks have a single defined outcome and a single expected process. Semi-open tasks have a single defined outcome, but no single expected process. The researchers created four tasks combining the two options for each dimension and presented these to all participants for completion. The study concluded that semi-open tasks assess a greater number of computational concepts and processes, offer more levels of difficulty, and provide better discrimination among participants than closed tasks. The directionality of the task did not have an effect on student scoring in their study. That is, asking participants to troubleshoot code was as effective a tool as having them create code (Zhong et al., 2016).

Design scenarios with semi-open tasks are a typical method in most literature on CT assessment (Lye & Koh, 2014), and require qualitative analysis of participants'

responses and programming code. Qualitative instruments are effective with small sets of participants to provide the researcher or instructor with insight into learners' retention of concepts and performance of the practices. However, design scenario assessments can be labor-intensive and so do not scale well to larger populations. Additionally, the validity and reliability of scenarios are sensitive to influence by other factors. For example, the degree of difficulty presented by a scenario may be perceived differently by participants than by the researcher (McCracken et al., 2001). Also, lessening the cognitive load on novice programming students by providing greater access to explicit knowledge regarding the scenario and the computational concepts, such as language reference manuals, has been shown to improve students' performance on computational practices (McCartney, Boustedt, Eckerdal, Sanders, & Zander, 2013).

Post-secondary Experiences with Programming Competencies

The programming competencies now being advocated as learning objectives for computational thinking in primary and secondary schools have been the subject of instruction at the higher-education level for more than four decades. Subsequently there is a body of research into student and faculty experiences in computer science education at the post-secondary level that pre-dates definition of competencies for CT. This section relates this body of research into programming competencies at the post-secondary level with the prior section on competencies for computational thinking.

Students are often surprised at the level of difficulty they encounter in the introductory programming courses (Robins, Rountree, & Rountree, 2003). The study of programming imposes a significant cognitive load on students (Garner, 2009). Programming requires the learner to quickly assimilate numerous new skills, concepts,

and details and then be able to apply this new knowledge in demonstrations that are without error. Studies have shown it to be a complex cognitive task (Guzdial & Morrison, 2016). Many students are not accustomed to the levels of precision and accuracy required by the programming languages they study. Since the computer demands perfection in the code they write, students often fail in their coding attempts. Successfully completing an assignment requires persevering through repeated failures which can be perceived as a frustrating and time-consuming experience (Campbell, 2016). Repeated unsuccessful attempts can seem like personal failures and can take a toll on students' self-efficacy and motivation (Fang, 2012; Garner, 2009). Consequently, high rates of course drop-outs and failures are common (Bergin & Reilly, 2005; Caspersen & Kolling, 2009).

Among computer science faculty, a common perception is that not all students are capable of learning to program (Lewis, 2007). This perception has been called the Geek Gene hypothesis (McCartney, Boustedt, Eckerdal, Sanders, & Zander, 2017) which states that based on innate ability, there are two types of students (a) those who can learn to program, and (b) those who can't. The existence of these two categories is argued to be reflected in the frequent occurrence of bimodal grade distributions in introductory programming courses (Dehnadi & Bornat, 2006).

One empirical study was highly influential (McCartney et al., 2013) in reinforcing this perception. The McCracken working group (MWG) sought to assess university students' basic mastery of fundamental programming skills at the conclusion of their first year of computer science course work (McCracken et al., 2001). This research objective was based on a predecessor to ABET student outcome 2. The McCracken study presented participants ($n = 216$) from four universities in the United States, Australia, and Europe

with a short programming challenge to create a calculator application. Students were given a fixed timeframe during a lab period to read through the challenge and related material; design a computational solution; and write the code to implement the solution. During the lab period the participants were not allowed access to other materials such as textbooks or language reference guides. This is comparable to a forward direction, semi-open design scenario as in Zhong et al. (2016). The students' performance at the task was assessed based upon the execution of their solutions in a black-box test, an examination of their code for style, and an examination of the code for closeness to a correct solution. Most students' performance fell far below instructors' expectations for the learning outcomes of their first-year students. Many of the programs were incomplete and either would not compile, or could not execute to normal completion. Examination of the designs and the program code indicated that the majority of students lacked a viable design or a complete solution. Additionally, the assessed scores had a bi-modal distribution (McCracken et al., 2001).

Within a decade of the McCracken study, many computer science educators had come to discount its findings (Lister, 2011) due to several issues with its methodology and a belief that the programming challenge was too difficult. Therefore, a new study was conducted to replicate the objectives and approach of the MWG with corrections to address its problematic issues (McCartney et al., 2013). In this new study the original programming task was modified to lessen the cognitive load by removing some problematic knowledge requirements and by providing open access to relevant documentation including online language reference guides. Using the same scoring

criteria and scale as the MWG, mean scores in the new study improved to 68 out of 110 possible from the prior mean of 23 in the McCracken study (McCartney et al., 2013).

More recent studies have sought empirical evidence for either the geek gene or the prevalence of bimodal grade distributions and have generally refuted both. Other factors that may explain extreme distributions in the grades have been presented by Robins (2010), Ahadi and Lister (2013), and Patitsas, Berlin, Craig, and Easterbrook (2016). However, the geek gene hypothesis still influences computer science instructors as a folk pedagogy. The persistence of this perception is significant as it represents a fixed mindset and presents a risk of self-fulfillment (McCartney et al., 2017).

Computational Practices as Acquired Skills

Aho's (2012) definition of CT categorizes both computational concepts and practices, as does the framework of Brennan & Resnick (2012). In a recent essay, Denning (2017) argued that evaluations of attainment of computational thinking too frequently depend on measures of knowledge retention of the computational concepts. He asserts that it is possible for students to perform well on such tests yet still lack the ability to transfer that knowledge into computational practices. He reasoned that these computational practices are cognitive skills, and that skills only develop over time as the learner gains experience. As an alternative form of assessment, he contended that students' competency with computational practices be evaluated in the manner of skills competency.

In order to assess computational thinking competencies as skills, a model of skill development is required. Denning recommended the Dreyfus model of directed skill acquisition (S. E. Dreyfus & Dreyfus, 1980). According to this model, skill development

begins as rule-based behaviors in the novice, and progresses to situation-based behaviors which are complex, nuanced, and intuitive when demonstrated by the actions of experts. Denning concluded his thoughts on assessing CT development by stating that “We need guidelines for different skill levels of computational thinking to support competency tests” (Denning, 2017, p. 36). That is, an assessment of a learner’s CT abilities should be located on a scale of CT skill levels, such as the stages of the Dreyfus model.

This model of skill acquisition relates changes in cognitive processes to learner progressions in experience and ability through five stages from novice to expert (H. L. Dreyfus, Dreyfus, & Zadeh, 1987; S. E. Dreyfus & Dreyfus, 1980). The model has been widely adopted for education of healthcare professionals (Carraccio, Benson, Nixon, & Derstine, 2008). However, for each field of expertise, the model requires interpretation of the characteristics displayed by learner’s at each stage. Benner (1982) adapted the model for development of nursing skills (Peña, 2010). Carraccio et al. developed an interpretation for medical students and residents in clinical practice skills. By combining this model with Kelly’s personal construct theory (1955) this paper has provided a framework that describes the stages of development for problem-solving computational practice skills.

Attaining Problem-solving Competency

Competencies for problem-solving skills are a requirement in many fields of higher education. Studies of the development and assessment of such conceptual expertise have been based on comparisons between novices and experts in fields of science such as physics, biology, and chemistry. By examining differences in the problem-solving processes of the two groups, insights can be gained into the cognitive

structures and processes available to each. If consistent differences can be reliably elicited and identified, then these differences can serve as markers of, and proxy measures for progress toward problem-solving competency.

Comparisons of the problem-solving abilities of novices and experts trace back to a set of studies into the categorization and representation of physics problems conducted by Chi, Feltovich, and Glaser (1981). Earlier research had established that problem-solvers use knowledge schemata to construct cognitive representations of problem descriptions, and that success at solving problems is dependent upon the quality of these problem representations (Hayes and Simon, 1976; Newell and Simon, 1972; Simon and Simon, 1978). Chi et al. (1981) investigated the interaction between these cognitive representations of problems and relevant knowledge by hypothesizing that (a) creation of a problem representation involves a categorization task, (b) categorization of the problem triggers associated knowledge in the solver's knowledge base, (c) and the construction of the problem representation may be constrained by the available context of this associated knowledge. Therefore, any differences in problem-solving performance between novices and experts should be attributable to deficiencies in the novices' available categories of domain knowledge which in turn, limit the construction of problem representations sufficient for an effective, efficient, and expedient problem-solving process.

Chi et al. investigated the process of problem categorization by novices and experts with open card sorts and interviews. The researchers confirmed their hypotheses. Experts and novices categorize the same problems differently. The categories of novices tend to be based on explicit, surface cues found in the problem description. Experts, however, perceive more information in a problem statement than do novices. Experts

have a great deal of tacit knowledge available for making inferences and derivations from the problem statement. Their recognition of cues and associated significance within a problem statement are based on the context or situation, and not just lexical meanings. Expert schemata are organized by derived factors and contain both declarative knowledge, used for categorization, and procedural knowledge used for taking action. The elicited procedural knowledge includes potential solution methods as well as features used to confirm, reject, or choose among these potential solutions. Thus, for experts, there is robust cognitive interplay between the knowledge structure and the problem representation (Chi et al., 1981).

The Chi et al. study used two card sort instruments. The first used 24 physics problems selected from a standard textbook. The elicited categorizations identified a significant difference between novices and experts. In order to confirm the interpretation of the difference as the result of using surface or deep factors as cues, a second card sort used 20 physics problems specifically constructed by the researchers to roughly pair six explicit surface factors with three implicit physics principles. These implicit factors were hypothesized by the researchers to represent the categorization cues used by experts. The results of the second sort confirmed this conclusion (Chi et al., 1981).

Subsequent studies of problem-solving expertise in other fields of science have followed this approach of using problems that combine hypothesized surface and deep factors in the stimuli set of an open sort. One recent study of undergraduate biology students used a stimulus set of 16 problems constructed as pairings of four hypothesized surface factors with four hypothesized deep factors. The deep factors were core concepts obtained from standard biology references. Novices were predicted to create four

categories corresponding to the surface factors with four stimuli each, while the experts were predicted to create four categories corresponding to the deep factors (Bissonnette et al., 2017). A similar instrument was used to research undergraduate chemistry students where problems were also constructed to pair four hypothesized surface factors with four hypothesized deep factors (Krieter, Julius, Bush, Scott, & Tanner, 2016). Another chemistry study used pairings of three hypothesized surface factors with three hypothesized deep factors (Irby et al., 2016). Participants in these studies completed only one open, or unframed, sort of the stimuli set.

For data analysis, the sorts in each of these three studies were quantified by counting the pairs of cards contained in each category. Thus, a category containing two cards would count as one pair (i.e., A-B), three cards would count as three pairs (i.e., A-B, A-C, B-C), while a category with four cards would count as six pairs. Expected pairings for the novice and expert categorizations were also identified. The sorts produced by experts and novices could then be compared by calculating the percentage of expected pairings for novice or expert classifications found in participants' sorts (Bissonnette et al., 2017; Irby et al., 2016; Krieter et al., 2016). Experts were found to have high percentages of the expected deep factor pairings. Novices, as expected, had few deep factor pairings, and many hypothesized surface factor pairings. However, the novices also had many unexpected card pairings (Bissonnette et al., 2017; Krieter et al., 2016). Unexpected card pairings indicate a surface or superficial grouping criteria by a participant that the researchers had not foreseen. These studies were able to conclude that card sort instruments are able to reliably differentiate between the problem-solving competency of novices and experts; and that experts categorize their problem

representations based upon deep factors. They also concluded that novices seldom use deep factors for categorization, and based upon the number of unexpected pairings chosen by novices, the surface factor categorizations of novices are less predictable (Bissonnette et al., 2017; Krieter et al., 2016).

The Krieter et al. (2016) study in chemistry also used two other quantitative measures for analysis: (a) edit distance, and (b) comparison-based index. Edit distance quantifies the structural similarity of two sorts (Deibel et al., 2005). By using the expected sort for the hypothesized deep factor categorization, and that for the surface factor categorization sort, the edit distance of each participant sort to these two expected sorts was calculated. The results were displayed as box plots of the distance between sorts by novices and experts to the hypothesized sorts of surface factors and that of deep factors. These results corresponded with the percentage pairing statistics (Krieter et al., 2016).

The comparison-based index for a sort is based on the frequency of observed pairings in both the novice and the expert sorts. The comparison-based index is derived by creating a comparison frequency matrix that subtracts the frequency count for each pairing observed for novices from the frequency count of the same pairing observed for experts. Thus, if a specific pairing is observed more frequently among the experts, the frequency matrix value for that pairing will be positive. Likewise, it will be negative for pairings more frequently observed among novices. The comparison-based index for a given participant's sort is the sum of the frequency matrix values for all pairings in the sort (Krieter et al., 2016). The researchers developed this index as a measure to be independent of the hypothesized sorts. That is, the results consider how all the

participants actually categorized the stimuli, as opposed to the researchers' preconceptions about the categorization.

The comparison-based index also provided a method to assess the development of problem-solving expertise in chemistry as students progressed through the undergraduate curriculum. The sort instrument was additionally administered to first-year majors, second-year majors, and upper division majors. Comparison-based index values for these sorts were calculated on the basis of the prior novice and expert frequency matrices. Box plots of the range of values for each cross-section demonstrated a monotonic trend toward expert-like sorting over time, with the largest movement occurring between the first and second years. The upper division group indicated continued movement toward expert-like, but the increase was not statistically significant.

Problem-solving Competencies in Computer Science

A search of the recent computer science education literature does not disclose similar studies regarding development of problem-solving expertise. However, four articles (Deibel et al., 2005; Fossum & Haller, 2005; McCauley et al., 2005; Sanders et al., 2005) were found that utilized card sorts to investigate the conceptual structures of computer science undergraduates regarding programming concepts. These articles share a common origin in a National Science Foundation funded initiative for bootstrapping research in computer science education (Sanders et al., 2005). The initial study by Sanders et al. (2005) focused on students who were completing their introductory courses in programming. The study design followed a constructivist perspective in order to understand how the students create meanings for basic programming concepts and the relationships among the concepts. The sample was drawn from 22 institutions, six

nations, and involved 23 researchers. Due to this large, diverse, and distributed population of participants ($n = 276$) and the variability of factors such as programming languages and instructional practices utilized across the institutions, the researchers concluded that traditional qualitative methods of data collection were impractical (Sanders et al., 2005). They instead chose to use a repeated single-criterion open card sort as their data collection method as this instrument is participant-focused rather than researcher focused and can be administered in a consistent manner across diverse segments of a population (Sanders et al., 2005).

In contrast to the card sort instruments and research designs employed for the studies of problem-solving expertise in physics, biology, and chemistry, the data collection instrument used with these computer science students consisted of single word prompts for programming concepts rather than problem scenarios. Additionally, while Bissonnette et al. (2017) and Krieter et al. (2016) constructed their stimuli sets as combinations of hypothesized surface and deep factors selected from generally accepted core concepts, Sanders et al. (2005) lacked a similar theoretical basis for such hypotheses. Indeed, an objective of this study was to gain insight into what the surface and deep factors might be. Therefore, a stimuli set was developed with prompts including 26 fundamental programming terms. To construct this list, the researchers consulted programming textbooks, academic literature on programming taxonomies, programming experts and computer science educators. The instrument was evaluated with a seven-person pilot study (Sanders et al., 2005).

Data collection utilized a physical card sort. Each participant was given a shuffled deck of 26 index cards, with one of the programming terms on each card, and asked to

sort the cards into categories of their own choosing using a single criterion for the sort. For each sort the participant supplied a label for each group and a description of the sorting criterion. These labels and descriptions were recorded by the researchers along with the cards making up each group. Upon completion of a sort the cards were collected back into a deck and reshuffled. Each participant was asked to repeatedly sort the cards using a new criterion until the participant was unable or unwilling to continue (Sanders et al., 2005). The repetition of the sorting task is another difference between this data collection design and those for the biology and chemistry studies. In order to gain the greatest insight into a participant's knowledge structure, a collection of sorts from that participant is required (Rugg & McGeorge, 2005). This data collection activity resulted in an information-rich dataset of 1260 sorts with a total of 5053 recorded categories (Sanders et al., 2005).

Sanders et al. (2005) lacked hypothesized sorts for surface and deep factors for the programming concepts, and did not use a quantitative analysis of frequencies of card pairings. Instead, they utilized qualitative methods to analyze the participants' supplied group labels and sort criteria descriptions. These data were aggregated across the sample with a hierarchical cluster analysis on both a verbatim basis and on the basis of an interpretation of the meaning, that is, a gist analysis. A gist analysis is a qualitative technique that aggregates items with a similar meaning despite use of different words, such as loop, iterative, and repetition. However, the analysis was constrained by the quantity of data and the lack of automated tools for lexical analysis (Sanders et al., 2005).

Based on this constrained, qualitative analysis of the collection of sorts, Sanders et al. were able to conclude that despite the diversity of institutions, instructional

methods, programming languages, and levels of experience there was a clear similarity across the population between the top ten sort categories. This indicated a consistent development of conceptual structures regardless of programming language or institution. The researchers also suggested that additional information could be gained from the dataset if new techniques for analysis could be developed which are more amenable to automation (Sanders et al., 2005).

Three groups of researchers from the Sanders et al. (2005) study pursued the challenge to create new computational techniques for attributing meaning to card sort data (Fincher & Tenenberg, 2005). These efforts resulted in the card sort measures for edit distance (Deibel et al., 2005) and orthogonality (Fossum & Haller, 2005). Edit distance compares the structural similarity of two sorts. Orthogonality expresses the dissimilarity of sorts within a collection. The edit distance is one of the statistics calculated by the more recent studies of Bissonnette et al. (2017) and Krieter et al. (2016) in their analyses. Fossum and Haller (2005) and McCauley et al. (2005) used the edit distance and orthogonality measures to further analyze the dataset from the Sanders et al. (2005) study. Both of these articles also collected additional card sort data from a sample of graduating computer science seniors ($n = 65$). This collection process used the same stimuli set as Sanders et al. (2005) and so enabled a statistical comparison of differences between the cognitive structures of novice programming students and those of graduating seniors.

McCauley et al. (2005) found that the orthogonality of sort collections of graduating seniors correlates to their categorization by quartiles when ranked by academic performance in their computer science courses. This trend in orthogonality was

shown to be statistically significant. The best performing seniors produced sorts with the greatest degree of differentiation. This finding is in keeping with the Chi et al. (1981) conclusion that the cognitive structures of experts contain greater degrees of differentiation than those of novices, as evidenced by the orthogonality of their sort categorizations.

Using the same ranking and categorization by quartiles of the graduating seniors, Fossum and Haller (2005) found that the orthogonality of the sorts produced by all but the lowest quartile of graduating seniors showed a statistically significant difference from the orthogonality of sorts produced by all novices. Therefore, the orthogonality measure can be used to reliably distinguish between the card sorts produced by novices from those produced by students with more developed problem-solving expertise. Furthermore, the sorts of the lowest quartile of seniors showed no statistically significant difference in orthogonality from that of the novice sorts (Fossum & Haller, 2005). This is an indication that the lowest quartile of graduating seniors failed to develop their problem-solving expertise beyond that of introductory programming students.

Since McCauley et al. (2005) identified that significant differences exist between the sorts of the highest and lowest performing graduating seniors based upon their respective orthogonality, they attempted to interpret the sorts of these two groups to identify how the changes in differentiation manifested. Using a qualitative content analysis of the criterion and category names provided during the sorts by each participating senior, a set of 16 representative content analysis groups (CAGs) were derived and used as the basis for coding the 291 sorts. The frequency distribution of the sorts within these CAGs was then analyzed by student performance quartile. The results

identified a difference in the frequency of selection of the CAGs between the highest and lowest performing quartiles (McCauley et al., 2005). This technique is analogous to the comparison-based index utilized by Krieter et al. (2016) to evaluate sorts without comparison against hypothesized sorts, although the methods of collection and analysis differ significantly.

Interestingly, for the majority of the identified CAGs, the calculated orthogonality confirmed that the sorts selected for each CAG were structurally similar. Thus, the derivation of the CAGS was validated both qualitatively and quantitatively. Additionally, the edit distance metric was calculated for each pair of sorts within a CAG. For each CAG, this process identified one sort which had the lowest edit distance to all the other sorts in that CAG. That is, the sort that was most structurally like all the other sorts in the CAG was identified quantitatively by its aggregate edit distance score. That sort was labeled as the structural exemplar sort for that CAG (McCauley et al., 2005). Since the edit distance and orthogonality metrics ignore the labels for categories and criteria, these exemplars provided a means to re-associate meaningful descriptors to each CAG. The exemplars derived from the collection of sorts in the quartile with the highest measure of orthogonality can also be considered to represent deep factors for categorization. Likewise, the exemplar sorts derived from the collection of sorts of the novice programmers could represent the surface factors.

Summary

Competency requirements for learner attainment of ABET student outcome 2 have most recently been researched and debated in the context of computational thinking (CT). Computational thinking is best defined as the set of thought processes used to

formulate a problem such that it can be solved computationally within a particular model of computation (Aho, 2012; Czerkawski & Lyman, 2015). Thus, operationalizing CT instruction and assessment must address (a) knowledge retention of the computational concepts of the model, and (b) acquisition of a set of computational practices for knowledge transfer with the purpose of problem-solving. This chapter has focused on the research conducted towards assessment of computational practices.

Two approaches for assessment of computational practices have been reviewed. The first is to require performance of the practices, and the second is to elicit categorizations of problem-solving expertise. Instruments for evaluating demonstrations of computational practices frequently provide a design scenario and request the learner to respond with either a forward directed activity, such as create the code, or a reverse directed activity, such as debug the code (Lye & Koh, 2014; Zhong et al., 2016). These assessments are often open or semi-open tasks requiring an element of qualitative evaluation. Studies that have utilized this approach to assess the programming competencies of post-secondary computer science students have had the reliability and validity of these instruments subsequently challenged. For example the McCracken (2001) task was judged as too difficult (Lister, 2011), and results were significantly improved with actions to reduce the cognitive load on the participants (McCartney et al., 2013). Multiple other factors have been investigated (Ahadi & Lister, 2013; Patitsas et al., 2016; Robins, 2010) for their influence on the scores of such evaluations and the common occurrence of extreme distributions that result.

An alternative approach for assessment of computational practices is based upon studies of differences in the cognitive representations and processes of experts as

compared to novices for problem-solving (Chi et al., 1981). Recent studies of problem-solving skill competencies in the fields of chemistry and biology (Bissonnette et al., 2017; Irby et al., 2016; Krieter et al., 2016) have employed and validated this form of assessment. A pair of earlier studies (Fossum & Haller, 2005; McCauley et al., 2005) in computer science also evaluated and refined techniques for this approach. Each of these studies confirmed that while the novices approach a problem-solving task by taking cues from explicit, surface factors stated in a problem description, the more skilled senior students, like their faculty, are guided by deeper factors that they derive from the problem statements. For the chemistry and biology studies, the deep factors were accurately hypothesized and one of the studies (Krieter et al., 2016) found evidence of a progression of development toward these deep factors in students during their undergraduate years. However, there is a gap in comparable research for computer science students as the representations of associated deep factors for computational practices have not yet been identified and the progression of their development toward these factors has not been investigated using knowledge elicitation methods.

Furthermore, the only two studies of differences in the cognitive representations between computer science novices and seniors (Fossum & Haller, 2005; McCauley et al., 2005) concluded that this cognitive ability for computational practices does not develop for all graduating seniors. Therefore, development of this ability must be influenced by factors other than just the time and exposure to the curriculum which all seniors share in common. The geek gene hypothesis states that this difference is due to innate ability. However, the Dreyfus model of skill acquisition states that development is facilitated with repeated practice and application of the skill. An extensive search of literature has

found no additional studies using knowledge elicitation techniques to research factors that influence the progression of development of the ability of computer science students to attain ABET student outcome 2.

Therefore, this review of the literature has identified two gaps in the research of computer science education that need to be pursued. The first gap is the lack of follow-up to study results (Fossum & Haller, 2005; McCauley et al., 2005) indicating that the computational practices of the lowest performing quartile of graduating seniors did not develop significantly beyond the skill level of novices, while the top quartile of graduating seniors did demonstrate a statistically significant growth in their computational practices skills. This difference should be investigated to determine if it is the result of innate ability or if it is influenced by accumulated programming experience in accordance with the model of skill acquisition.

The second gap is the lack of recent studies (Fossum & Haller, 2005; McCauley et al., 2005) in computer science education comparable to that for chemistry education (Krieter et al., 2016). Differences in the cognitive representations of computer science novices and seniors were last investigated in 2005. Krieter et al. (2016) very recently investigated similar differences for chemistry students and charted development toward expert-like skills as students progressed through the curriculum. No study has been found in the literature for computer science education that investigates the progression of skill development as these students progress through the curriculum.

This study proposed to address these gaps in three steps. First, it replicated the previous studies of conceptualizations of computer science students to determine if similar differences in categorizations exist within the cross-section of completing

students at the participating institution according to their academic performance. Second, it investigated how the conceptualizations of those students most likely to have attained the desired level of programming skill differ from those of the least likely. Finally, it analyzed how development toward conceptual expertise is affected by students' levels of instruction and programming experience as they progress through the curriculum.

The following research questions were addressed in this study:

1. Is there a relationship between categories of coursework achievement (Introductory, Mid-Program, Completing) and categories (High, Low, or Zero) of card sort orthogonality?
2. How many reliable and interpretable structural exemplar sorts are derivable from the collections of sorts produced by those participants most likely to have attained the desired level of programming skill?
3. Is there a statistically significant difference among the categories of computer science students' progression through milestones of coursework attainment (Introductory, Completing, and Mid-program) on the dependent variable, the edit distance between the card sorts to the exemplar sorts?
4. Is there a statistically significant difference among the categories of computer science students' programming experience (Light, Moderate, Extensive) on the dependent variable, the edit distance between the card sorts to the exemplar sorts?

CHAPTER III

Methodology

The purpose of this study was to establish a baseline of quantitative measures of computational thinking skill acquisition as an aid in evaluating student outcomes for programming competency. Proxy measures for the desired skill levels were identified that reliably differentiate the conceptual representations of computer science students most likely, from those least likely, to have attained the desired level of programming skill. Insights about the development of computational thinking skills across the degree program were gained by analyzing variances between these proxy measures and the conceptual representations of cross-sections of participating students partitioned by levels of coursework attainment, programming experience, and academic performance. Going forward, similar measures can provide a basis for quantitative assessment of individual attainment of the desired learning outcome.

The study used the knowledge elicitation instrument (a repeated, open card sort of 26 programming terms) and quantitative methods from McCauley et al. (2005) and Fossom and Haller (2005). The theoretical framework hypothesized that learners' organization of knowledge changes as learners gain experience with a skill over time (S. E. Dreyfus & Dreyfus, 1980), and that these changes are reflected in the elicited conceptual representations (Kelly, 1955). Therefore, cross-sections of students in the computer science degree program at the participating institution were selected for elicitation of conceptual representations reflective of their attainment of specific milestones of coursework, level of programming experience, and category of academic performance. A cross-sectional analysis approach similar to that used by Krieter et al.

(2016) and Bissonnette et al. (2017) compared differences between groups to identify differentiating indicators of progression in computational thinking skill acquisition across the degree program. This chapter describes the design of the research, and the methodology to be used to conduct the study and to analyze the collected data.

Research Questions

The following research questions were addressed in this study:

1. Is there a relationship between categories of coursework achievement (Introductory, Mid-Program, Completing) and categories (High, Low, or Zero) of card sort orthogonality?
2. How many reliable and interpretable structural exemplar sorts are derivable from the collections of sorts produced by those participants most likely to have attained the desired level of programming skill?
3. Is there a statistically significant difference among the categories of computer science students' progression through milestones of coursework attainment (Introductory, Completing, and Mid-program) on the dependent variable, the edit distance between the card sorts to the exemplar sorts?
4. Is there a statistically significant difference among the categories of computer science students' programming experience (Light, Moderate, Extensive) on the dependent variable, the edit distance between the card sorts to the exemplar sorts?

Null Hypotheses

- H₀₁*: There is no statistically significant relationship between categories of coursework achievement (Introductory, Mid-Program, Completing) and categories (High, Low, or Zero) of card sort orthogonality.
- H₀₂*: No reliable and interpretable structural exemplar sorts are derivable from the collections of sorts produced by those participants most likely to have attained the desired level of programming skill.
- H₀₃*: There are no statistically significant differences among the categories of computer science students' progression through milestones of coursework attainment (Introductory, Completing, and Mid-program) on the dependent variable, the edit distance between the card sorts to the exemplar sorts.
- H₀₄*: There are no statistically significant differences among the categories of computer science students' programming experience (Light, Moderate, Extensive) on the dependent variable, the edit distance between the card sorts to the exemplar sorts.

Research Design

In order to investigate and answer the research questions it was necessary to gather descriptive statistical data from computer science students regarding the level of development of their conceptualizations of programming as they progress through their computer science degree program. A single point-in-time, cross-sectional, survey design was chosen for this purpose. From the population of all computer science students at the participating institution, cross-sections of the collected data were elicited for comparative

statistical analysis. The chief limitation of this approach is that it cannot determine causal relationships (Cohen, Manion, & Morrison, 2011).

Prior studies of the development of conceptual expertise have utilized the card sorting elicitation technique for data collection and analysis. In McCauley et al. (2005) significant differences between the top and bottom quartiles of academic performance in computer science seniors were found in their conceptualizations of programming. Krieter et al. (2016) identified changes in the way chemistry majors conceptualized problem solutions as they progressed through their coursework. The selection and use of methods for collecting and analyzing card sorting data to address the research questions of this current study were drawn from these and other card sorting studies. These additional research design details are provided in the following sections.

Selection of Participants

The population for this study was the set of students enrolled in computer science courses in the Department of Computer Science during the Spring 2019 semester ($N \sim 430$) at the participating institution. Most of these students take multiple computer science courses each semester, as evidenced by the current enrollments for the Spring classes, approximately 850, exceeding the number of individual students in the Computer Science department. Therefore, the sampling procedure was designed to efficiently solicit a majority of computer science students through a number of selected courses, quickly identify redundant solicitations to the same students, and prevent more than one data collection per participant.

The instructors for the courses listed in Table 1 were asked to make a study recruitment announcement to their classes. Recruitment materials were provided online to

inform students about the nature and purpose of the study and to provide video examples of card sorting tasks. Students were informed that participation would require an hour of their time. The students could then choose whether or not they wished to register for the study. Instructors were encouraged to award participation points to students who participated. Solicited students were informed of the nature of the research, its potential benefits and risks, and asked to consent. The principle risk to participants was a breach of confidentiality. To protect against this risk, a randomly generated participant identifier was assigned to each consenting participant. This identifier was used exclusively in the study to associate all collections and analyses of data to the participants.

Students who chose to participate provided their school e-mail address as part of an online registration activity. If students were solicited in more than one class, they only needed to register one time. However, the cloud-based application that received these submissions identified redundant registrations from the same e-mail address and if found, just thanked the student for their request.

The courses listed in Table 1 were selected to provide adequate solicitation for the various cross-sections of students required to address the research questions. Student progression through the degree programs was categorized into five stages of milestone achievement. First were introductory students in course A, 1436. Last were graduating seniors (E) who had completed D level courses, and had 28 or more hours of computer science coursework with a cumulative GPA of at least 2.0. The intermediate achievement milestones were determined by current enrollment in, or completion of, course B, 1437, 2327, or DFSC 1316; course C, 2329, 3318, or 3327; and course D, 3319, 4318, or 4319.

Table 1

Computer Science Courses to be Solicited for Student Participation in the Study

	Course Number	Description	Spring Enrollment
A	1436	Fundamentals I	100
B	1437	Fundamentals II	60
	2327	Networking Intro	55
	DFSC 1316	Digital Forensics I	25
C	2329	Machine Language	55
	3318	Databases	55
	3327	Computer Architecture	55
D	3319	Data Structures	55
	4318	Advance Language Concepts	44
	4319	Software Engineering	28
	4326	Network Theory	30
	4327	Operating Systems	31
	4349	Professionalism and Ethics	36
	DFSC 4317	Info Security	15

Table 2 provides details on the planning basis for soliciting cross-sections of students by milestone achievement. The number of students at each milestone could only be estimated. Based on these estimates, a desired sample size was interpolated from a table of sample sizes for categorical data by Bartlett et al. (as cited in Cohen et al., 2011, p. 148) given a desired alpha of 0.05 and a margin of error of 0.05. The total desired sample size of 338 for all cross sections exceeds the sample size of 305 calculated with

the G*Power software for a one-way ANOVA with five groups at an alpha of 0.05, statistical power level of 0.95, and a medium effect size of 0.25. Soliciting from all students in the Table 1 courses was anticipated to result in a greater number of solicitations than required for each desired sample size and to also exceed the total population, indicating that some students will receive multiple solicitations.

Table 2

Computer Science Cross-Section Details

Milestone	Estimated Size	Minimum Desired Sample Size	Soliciting
A	90	74	100
B	90	74	140
C	85	70	165
D	75	64	127
E (65	56	65)
Total	340	282	532

Instrumentation

The instrumentation for this study was based on that of a prior study. McCauley et al. (2005) conducted an investigative survey through use of a repeated single-criterion card sort (Rugg & McGeorge, 2005) regarding the conceptualizations that graduating computer science students have about programming. Their study also gathered background data for each participant including grades in all computer science courses, the computer science grade point average (GPA), and the overall grade point average along with demographic data. The computer science GPA was used to categorize students

into academic performance quartiles for purposes of comparison using several quantitative methods for statistical analysis.

This current study collected data with two online instruments for the participants, and by gathering official document artifacts from the institution regarding each participant. One online instrument solicited background and demographic information from the participants through a questionnaire. The other online instrument elicited the primary data for the study from the participants through a card sorting activity. Once the students completed both instruments their participation in the study was concluded. There were no follow-up questions or interviews with the researcher.

Card sort activity. Card sorting is a categorization task (Fincher & Tenenberg, 2005) where a participant sorts elements of a stimuli set into groups based upon a sorting criteria (Rugg & McGeorge, 2005). This active construction by the participant of the external groups, or categories, reflects their internal conceptualizations (Fincher & Tenenberg, 2005). Card sorts are especially well suited for capturing data of non-scalar or nominal categories, for which repertory grids are not suitable (Rugg & McGeorge, 2005). In an open, or unframed, type card sort, participants are allowed to choose the sort criteria, and to specify category labels. Thus, as a data collection method, open card sorts have the benefit of being participant-centric rather than researcher-centric (Fincher & Tenenberg, 2005). Administratively, card sorts have an advantage of being easy to use, relatively quick, and systematic (Rugg et al., 1992). For analysis, card sort results reveal participants' categorizations which can be compared to identify similarities and differentiations in participant conceptualizations (Rugg & McGeorge, 2005).

When comparing card sorts produced by experts against those of novices, it is noted that experts generally can find more criterion for uniquely grouping cards than do novices (Fossum & Haller, 2005). In order to allow this tendency to manifest, each participant needs to be encouraged to repeat the card sort multiple times. Hence the term, repeated single-criterion card sort. This repetition results in a collection of card sorts for each participant.

This study utilized the repeated single-criterion card sort instrument originally devised by Sanders et al. (2005) and subsequently used in the McCauley et al. (2005) study. As an open card sort, the instrument consists only of the 26 item non-scalar stimuli set shown in Figure 2. The original researchers selected single-word prompts for programming concepts. These prompts were gathered from textbooks, academic articles, educators, and experts. A pilot test of the instrument was conducted with seven participants (Sanders et al., 2005).

Function	If-then-else	State	Loop
Method	Boolean	Encapsulation	Expression
Procedure	Scope	Parameter	Tree
Dependency	List	Variable	Thread
Object	Recursion	Constant	Iteration
Decomposition	Choice	Type	Array
Abstraction			Event

Figure 2. Stimuli set consisting of 26 single word programming terms for card sort use.

An online tool (see Appendix A) randomized these items into one vertical list and displayed them all at once along the left side of the screen as shown in Figure 3.

Participants then dragged each item to the right and dropped it into a new or existing category, as shown in Figure 4. Participants could enter a description for each category.

When all the objects had been placed within a group, the data was recorded (i.e., category names, and objects within each category) and the stimuli set randomized again. In a repeated single-criterion sort the respondent is asked to categorize the stimuli again using a different criterion. The sorting task continues until the respondent is unable or unwilling to create a new sort.

Computational Thinking Skills *Card Sorting*

Programming Terms Sorted into these groups based on-

-
-
-
-
-
-
-
-
-
-
-
-
-
-
-
-
-
-
-
-
-
-
-
-
-
-

M. Earnest Research

Figure 3. Initial screen display of card sort task. Twenty-six programming terms listed on the left side of the screen are to be dragged to groups on the right side.

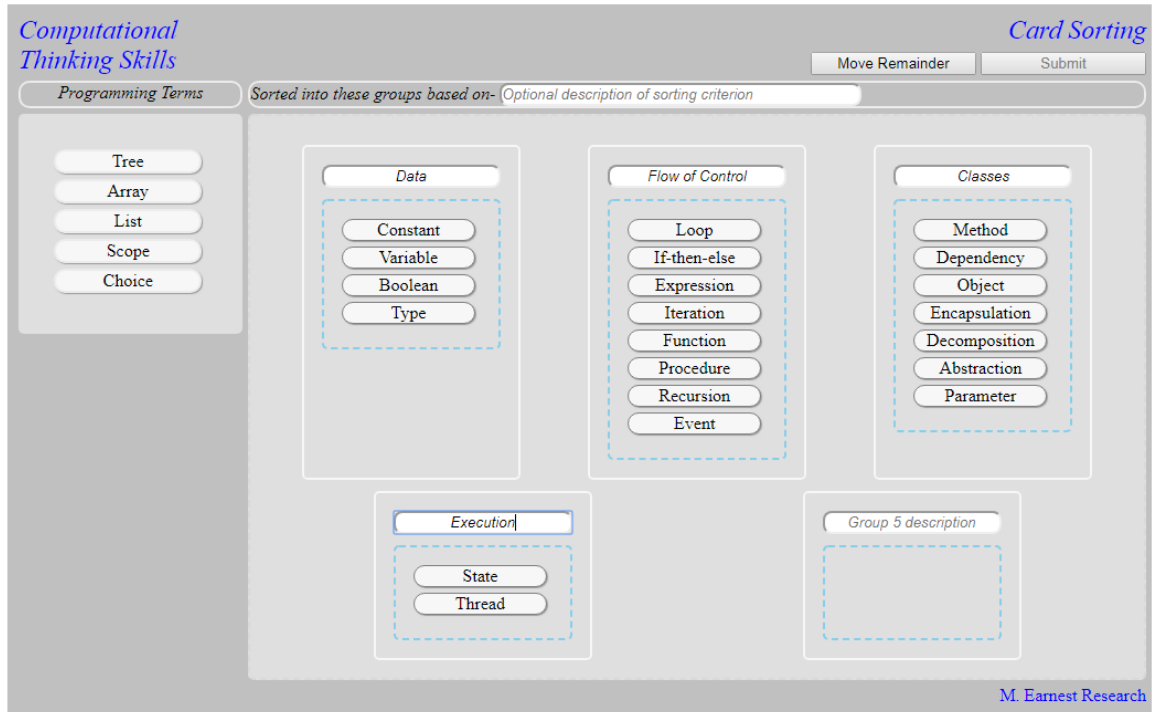


Figure 4. Card sort task underway. Four categories of data, flow of control, classes, and execution have been created with items from the stimuli-set dropped into them.

Questionnaire. An online questionnaire (see Appendix B) solicited background and demographic information from each participant. The demographic information collected was gender, age, ethnicity, intended major, and college classification level. Participants were allowed to supply their own descriptions for gender, and ethnicity, and these were subsequently categorized by the researcher. Background information solicited included questions regarding prior training or experience in programming, and Likert-scale evaluations of skill level for various types of programming activities. Answers to these background questions were used by the researcher to rank the participants according to their degree of programming experience. The ranked participants were then categorized into thirds (Light, Moderate, Extensive).

Document artifacts. For each consenting participant, university records were retrieved to collect the list of computer science courses taken each semester, and the

grades earned in each of these courses. The researcher calculated each participant's cumulative grade point average in their computer science coursework. Participants classified as completing students were ranked by the researcher according to their cumulative grade point average and categorized into thirds. Some other cross-sections of the participants were also ranked by cumulative GPA in computer science coursework for comparison against the completing students.

Validity and reliability. Card sort tasks are well established instruments for the investigation of expertise (Irby et al., 2016) and have been used in studies of physics (Chi et al., 1981), biology (Bissonnette et al., 2017), chemistry (Irby et al., 2016; Krieter et al., 2016), and computer science (Fossum & Haller, 2005; McCauley et al., 2005). The validity of a card sort instrument is dependent upon the stimuli set used to prompt the elicitation. The reliability of a card sort instrument is dependent upon the procedures used in the administration of the activity.

To improve the validity of the instruments for this study, the stimuli set used was the same one used by Sanders et al. (2005) to investigate systematic differences among a large and diverse population of 276 computer science students and instructors at 22 institutions in multiple nations with a resulting 1260 sorts. These stimuli prompts were originally selected from programming textbooks, articles on programming taxonomies, and recommendations from programming experts and educators, and then tested with a seven-person pilot group.

To improve the reliability of the instruments for this study, the instruments were delivered online and the procedures for administering the instruments were automated. The card sorting task followed the same procedures used previously with this stimuli set

(Fossum & Haller, 2005; McCauley et al., 2005; Murphy et al., 2005; Sanders et al., 2005) and as described in Rugg and McGeorge (2005). While accessing the document artifacts, the study followed University procedures for researcher access to student records. As all data was collected it was securely stored in a relational database. An online tool was developed to retrieve datasets for analysis in SPSS, and Excel based upon filters selected by the researcher. This tool also collected and journaled these extraction requests with date and time stamps to enable a reliable recreation of all datasets for statistical testing as needed.

Procedures

Preparation. The study was conducted with only a single researcher, and no outside funding. The population solicited for participation exceeded 325. Therefore, the administration of the instruments was conducted online and automated to the extent possible.

As the data was collected asynchronously, identification and authentication of each participant was essential for maintaining the integrity of the collected data. Each participant had to be properly identified and associated with only one study participant identifier. The study relied on the university assigned student email address for identification and authentication. A cloud-based application was developed to register participants upon receipt of an email address supplied by the student. The registration application associated a unique, random identifier, the participant identifier, with the submitted email address. This participant identifier was embedded in links returned to the student to open the collection instruments. Cloud-based applications were also developed

to receive these links, to authenticate the submitting participant, and then display the appropriate collection instrument.

Existing online tools for administering the questionnaire and card sort instruments were considered for use in the study. However, these tools had to be capable of being integrated within the authentication protocol of the study. This study imposed specific requirements on the card sort instrument.

The card sort tool had to support administration of repeated single-criterion sorts. Participants had to be able, and were encouraged to complete multiple, different categorizations of the same stimuli set. Participants also had to be required to assign all items in the stimuli set to a group, even if one group is labeled as “Don’t know”, or “Not applicable”. Due to these requirements, subsequent procedures for the card sort activity were written with the assumption of the instrument being specifically developed for this study.

The cloud-based application was developed in parallel with the Institutional Review Board (IRB) approval process. None of the data collection materials were made available to students until IRB approval of the study had been obtained. This approval occurred at the end of November. The timeframe for the data collection was targeted for the 6th week of the Sprint 2019 semester, February 20 through 26.

Collection. The procedures for data collection were presented to the population through announcements by their course instructors. The activity opened on February 20, and remained open for one week. Within this activity students were presented with an overview of the study, offered a demonstration of an example card sort activity, and informed as to their rights and risks associated with participating in the study. The

principle risk to participants was a breach of confidentiality regarding the data collected. They were given the right to decline to participate, or to withdraw from participation without consequence to their course grade.

Students who chose to participate indicated their consent by signing, dating, and submitting forms of informed consent and consent to the release of academic records. They then provided their email address in a registration form and submitted that form to the cloud-based registration application. The application verified the student's identity by sending a confirmation code to the email address. The student then entered the confirmation code into the registration page within a set timeframe. Upon confirmation of student identity, the application verified the uniqueness of the student registration request, and assigned each student a unique and random participant identifier. The student received a return page with a link, including an embedded participant identifier, to the questionnaire instrument.

The student used the link to open and complete the questionnaire regarding their demographic data and background information. The instrument collected and stored this data and associated it to the participant identifier. The student was then sent a second link, and the embedded participant identifier, to the card-sorting instrument. In this link, the student was asked to reserve 40 to 60 minutes to complete the task.

Students using the link to access the card sorting instrument were presented with the task instructions (see Appendix A), and given another opportunity to view a demonstration of a sample sort. When the participant was ready, the task began.

The instrument initially displays all 26 items of the stimuli set, as shown in Figure 2, listed down the left side of the window. Adjacent to the list of stimuli is one empty

group box. The participant drops a selected item into the group box and is prompted to enter a description for the group. At the top of the window a prompt opens requesting a description for the sort criteria. A button appears at the top that says “I am Done”.

The user then selects another item from the stimuli set. The outline of another empty group box appears either beside or below the existing group(s). The participant drops the item into one of the group boxes. This process repeats for each item in the stimuli list. At any time, the participant may move items between groups and may change the group labels.

At any time the user may click on the “Done” button. If items remain in the stimuli list, the participant is prompted that items remain to be sorted and asked if these items can be sorted. If the participant responds that they cannot be sorted, a prompt will ask to select a reason, either (a) don’t know this term, or (b) the sort criterion does not apply. When the participant selects one of these choices, a new group box appears with the selected description as the label, and containing the remaining items from the stimuli set.

When the list of stimuli items is emptied, the participant is prompted to provide any unnamed group labels. When the participant is finished with all edits, they click on the “Done” button. The tool then records the sort data, with a unique sort identifier, and displays an acknowledgement to the participant and prompts them to try to sort again, or to quit.

If the participant elects to sort again, the window returns to the initial state with the list of stimuli items randomized again. If the participant elects to quit and fewer than

four sorts have been completed, the participant is encouraged to try for just one more, and again given the choice to try again, or to quit.

Once the participant quits, the instrument closes. While the data collection period was open, if the participant used the link to return to the sorting instrument, they were allowed to complete additional sorts. If the participant creates a duplicate sort, this situation is not detected or prevented. Duplicate sorts become evident during data analysis.

Analysis. Each sort was assigned a unique identifier and tagged with the participant identifier of its creator. Through the participant identifier, each sort was also be tagged with the categorical information about that participant. Various collections of sorts were extractable from the database using any of these tags. A tool was developed to extract collections of sorts and save them as datasets suitable for analysis in SPSS, Excel, etc.

As mentioned above, demographic and background information collected with the questionnaire was reviewed by the researcher. Since some of this data is open ended, the researcher determined proper categorization. The researcher also collected institution artifacts regarding the participants and recorded appropriate data from those records. The data from these two sources was associated with the participant identifier and stored in the database prior to analysis activities.

Data Analysis

The objective in analyzing data collected from a card sorting instrument is to compare and contrast individual sorts and collections of sorts against each other. Such analysis provides insight into the differences in the participants' construction of their

conceptual representations (Fincher & Tenenberg, 2005; Rugg & McGeorge, 2005). A card sort activity collects both lexical and structural data. The lexical information consists of the labels the respondent provides for the categories, and the description of the sorting criterion. Traditionally, qualitative coding and clustering methods have been used to analyze the lexical data. However, card sort studies can easily generate volumes of data which are impractical for analysis with qualitative methods (Sanders et al., 2005). Quantitative measures for expressing the similarities and dissimilarities among card sorts, however, easily accommodate large sets of data and enable statistical methods of analysis to be applied in order to address research objectives (Deibel et al., 2005; Fincher & Tenenberg, 2005; Fossum & Haller, 2005). These quantitative measures ignore the lexical content and analyze only the structural data provided in the respondents' card sorts.

Measures. In order to explain the calculation and meaning of the quantitative measures of card sorts, it is first necessary to describe the structural information provided by the instrument. For the purpose of illustration, consider a case (Deibel et al., 2005) where a respondent produces two sorts, identified as A and B, from a stimuli set consisting of nine items, identified as 1 – 9. Sort A contains three categories: $A = \{A_1, A_2, A_3\}$ and sort B contains four categories: $B = \{B_1, B_2, B_3, B_4\}$. Furthermore, the nine stimuli objects are distributed across the categories in both sorts as shown in Table 3.

Table 3

Two Example Sorts

Sort A	Sort B
$A_1 = \{1, 2, 3\}$	$B_1 = \{1, 2\}$
$A_2 = \{4, 5, 6\}$	$B_2 = \{3, 4\}$
$A_3 = \{7, 8, 9\}$	$B_3 = \{5, 6, 7\}$
$A_4 = \{\}$	$B_4 = \{8, 9\}$

Note. Example sorts of a stimuli set of nine items. Sort A contains three categories and Sort B contains four categories (Deibel et al., 2005).

Table 3 represents the structural description of both sorts. In comparing these two sorts, the fundamental question is: how much alike are they? The answer to this question can be quantified as the edit distance metric.

It is also desirable to compare different collections of sorts. For example, how do the sorts produced by the top third of completing students compare to the sorts produced by the lowest third of completing students? Quantifying the orthogonality of each collection of sorts enables such statistical comparisons to be performed (Fossum & Haller, 2005).

Edit distance. The edit distance between two sorts is defined as the minimum number of card moves required to transform one sort into the other and is calculated as described in Deibel et al. (2005) with the following sequence.

Using the example in Table 3, the first step is to create an equal number of groups for both sorts. This is a requirement of the algorithmic solution. So we create category $A_4 = \{\}$, an empty group. Then, a minimum set of moves to transform A into B is the sequence: Move 3 from A_1 to A_4 ; move 4 from A_2 to A_4 ; and move 7 from A_3 to A_2 .

This results in two sorts as shown in Table 4. This allows the groups between the two sorts to be matched as: $A_1 = B_1$; $A_2 = B_3$; $A_3 = B_4$; and $A_4 = B_2$.

Table 4

Transformation of Sort A to be Identical to Sort B

Sort A	Sort B
$A_1 = \{1, 2\}$	$B_1 = \{1, 2\}$
$A_2 = \{5, 6, 7\}$	$B_2 = \{3, 4\}$
$A_3 = \{8, 9\}$	$B_3 = \{5, 6, 7\}$
$A_4 = \{3, 4\}$	$B_4 = \{8, 9\}$

Note. Sort A transformed to structurally match Sort B in terms of number of groups and contents of each group. Three item moves were required for this transformation (Deibel et al., 2005)

The edit distance calculation only counts the number of moves of individual stimuli items required for the transformation. The final reordering of the groups to achieve the final match is not considered. Therefore, in this example, the edit distance value is 3, and we can state that the two sorts differ by an edit distance of 3. The edit distance can be efficiently calculated using the Munkres, or Hungarian assignment algorithm for bipartite graphs (Deibel et al., 2005).

Note that the edit distance is a measure of structural similarity between two sorts. If the edit distance is zero, then the sorts are structurally identical, even if the category names assigned in both sorts are disparate, and possibly differ in meaning. The edit distance measure is a ratio scale metric (Deibel et al., 2005).

Orthogonality. The orthogonality of a collection of sorts is a measure of the degree of differentiation among the members of the collection. The measure is based on

the edit distance between every pair of sorts in the collection. For example, assume a collection contains four sorts J, K, L, and M. The distance between each pair of sorts can be calculated and represented as a matrix as in Table 5:

Table 5

Pair-wise Edit Distances Among Four Sorts.

	J	K	L	M
J	0	1	1	2
K	1	0	3	4
L	1	3	0	2
M	2	4	2	0

Note. Matrix of pair-wise edit distances among four sorts.

These distances and relationships can also be represented as a topological graph as shown in Figure 5:

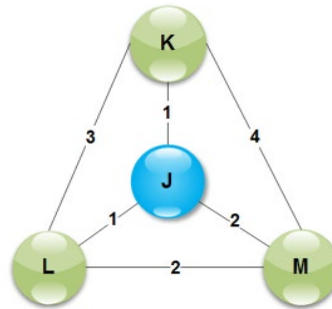


Figure 5. Example topological graph of a collection of four sorts with the edit distances between each pair of sorts displayed.

Each vertex in the graph represents one of the sorts within the collection, and each line (or *edge*) connecting a pair of vertices represents the edit distance (or *weight*) between those two sorts. A path through a topological graph along the edges that passes through all of the vertices exactly once is called a spanning tree. Note that not all of the

edges are included in any one spanning tree. For example, one spanning tree for the above example is $J \rightarrow L \rightarrow M \rightarrow K$ as denoted by the heavier edge lines in the left hand graph in Figure 6. The length of this path is the sum of the edge weights included in the path: $1 + 2 + 4 = 7$. Among possible spanning trees for a graph, the minimum spanning tree is the one with the shortest distance. In the example in the right hand graph in Figure 6, the minimum spanning tree is: $K \rightarrow J \rightarrow L \rightarrow M$ with a length of $1 + 1 + 2 = 4$.

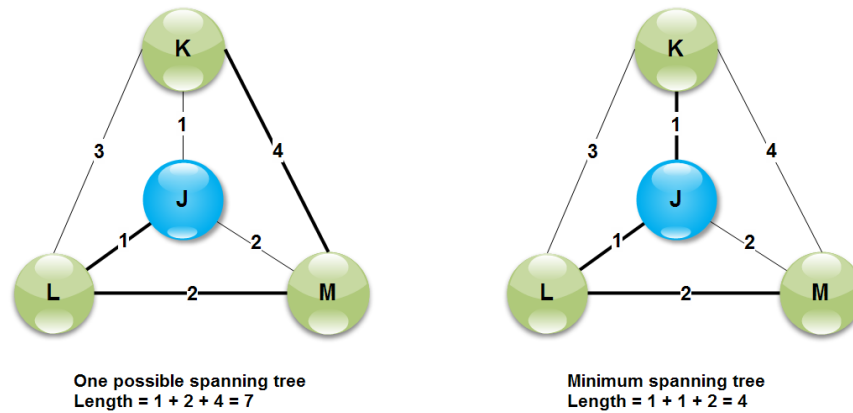


Figure 6. Illustration of minimum spanning tree.

The orthogonality measure of a collection of sorts is calculated as the sum of the weights of the minimum spanning tree divided by the total number of vertices in the graph (Fossum & Haller, 2005). This measure is also known as the normalized minimum spanning tree, or NMST. The need for this normalization is demonstrated by the following example. Consider the graphs in Figure 7 of two different collections where the minimum spanning tree of each is indicated with heavy lines for the edges. The length of the minimum spanning tree for both graphs is the same: 6. However, the graph on the right contains five sorts which exactly match each other structurally (edit distance = 0) and one sort at a large edit distance from the other sorts in the collection. In the graph on the left, all of the sorts are similar, but clearly differentiated from one another. That is, in

the graph on the left the sorts are more orthogonal than those in the graph on the right.

This distinction is reflected in the NMST values of the two graphs: 1.5 versus 1.0

(Fossum & Haller, 2005).

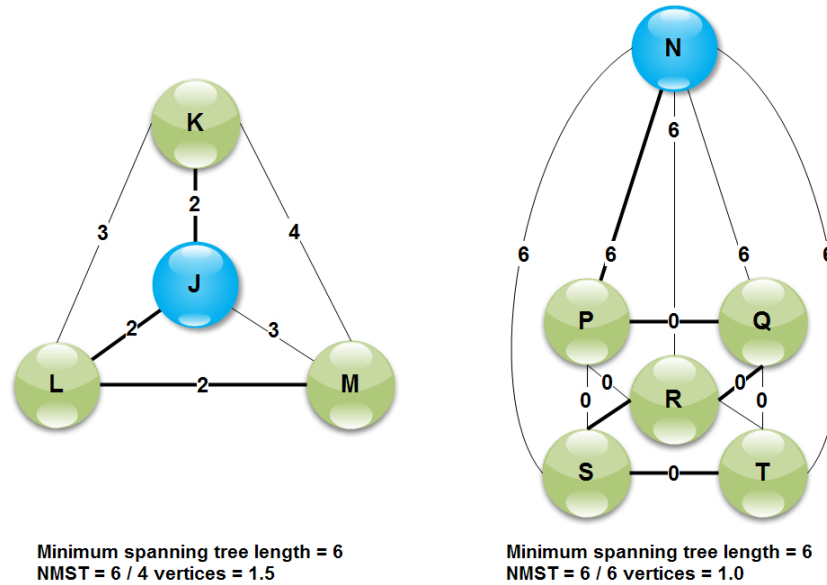


Figure 7. Greater differentiation of sorts represented by NMST measure.

Methods. The orthogonality measure, NMST, is an ordinal scale value; therefore non-parametric statistical tests are appropriate for its use in analytical methods. However, since the edit distance value is a ratio scale metric, it is suitable for parametric statistical tests. Additionally, the edit distance enables quantitative methods of analysis to identify clusters of sorts which may be embedded within a collection (Deibel et al., 2005; Precht et al., 2014). The edit distance also can be used to evaluate the proximity of a sort collection to a probe or exemplar sort (Deibel et al., 2005; Krieter et al., 2016).

Clusters. The edit distance can be used within a collection of sorts to identify clusters of similar sorts. Deibel et al. (2005) define a d -clique as the set of sorts that are all within a distance d of each other. For example, the collection shown in Table 5 and Figure 5 is a d -clique with a distance of 4, since the weight of each edge is less than or

equal to 4. Within a clique, the vertex with the shortest total edit distance to all of the other sorts is the structural exemplar sort (McCauley et al., 2005). In Figure 5, vertex J is the structural exemplar sort with a total edit distance to the other vertices of $1 + 1 + 2 = 4$. The exemplar sort is the most nearly equidistant sort within a cluster (McCauley et al., 2005).

Sorts within a cluster will have highly similar structural characteristics. However, this may not assure that this set of sorts have similar lexical meaning, depending upon the value of d which is selected for identification of the cluster. This study compared the lexical data of the sorts within a cluster before accepting the equivalence of the sorts. This was the reverse of the procedure followed by the McCauley et al. (2005) study which first performed a qualitative content analysis on the lexical data of all the collected sorts and then verified the structural similarity of the resulting clusters.

An algorithm for finding cliques of edit distance size d is discussed in Deibel et al. (2005) as well as in Precht et al. (2014). This study chose to use the Bron-Kerbosch algorithm for identification of cliques within an edit distance table where all values are d or less.

Probe proximity. The previous measures and methods allow for comparison among the sorts collected from the respondents of an open sorting activity. However, a researcher may have interest in comparing a collection of sorts against a particular sort criterion. The researcher may introduce this criterion as a probe, either as a hypothetical sort, or by identifying one sort as an exemplar (Deibel et al., 2005). The proximity of various clusters or collections to this probe sort allows for the collected data to be compared to the referenced criterion. The proximity measure can be taken as either the

mean of the set of edit distances between the probe sort and each of the sorts within the collection or as the smallest edit distance between the collection and the probe. This later technique can be graphically expressed as a box plot, suggested by Krieter et al. (2016), and represented in Figure 8 where the open sorts of novices and faculty are compared to a probe sort representing a hypothetical, cognitively complex categorization. In this example, the box plots clearly indicate that the sorts of the faculty have greater proximity to the probe sort, that is, are closer to an edit distance of zero, than are the sorts produced by novice students.

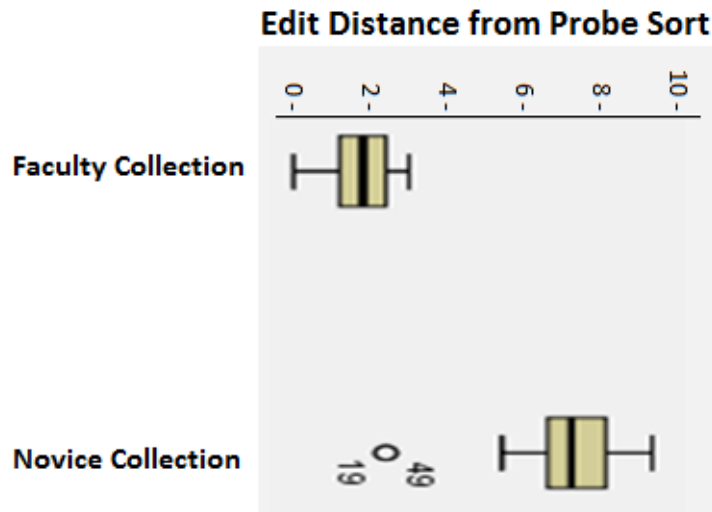


Figure 8. Box plot of sort proximity to a probe. The proximity, expressed as edit distance on the horizontal scale, of open sorts of novices and faculty to a probe sort representing a hypothetical, cognitively complex categorization.

Tools. SPSS was used for statistical analyses. Node.js applications were written by the researcher to implement the data collection procedures. Publicly available Node.js implementations of graph theory algorithms were used for derivation of edit distances, minimum spanning trees, and d -cliques. Example sort collections provided in Fossum and Haller (2005) were used to validate the accuracy of the edit distance and orthogonality implementations.

Summary

Computational thinking is becoming a learning objective at all levels of the educational system. However, developing learners' computational thinking requires more than growing individuals' accumulations of facts pertinent to programming. Instead, computational thinking is a skill developed over time through practice. Therefore, progression of learners' computational thinking abilities should be assessed against a baseline of expected indicators of skill acquisition over a period of practice time. However, no such baseline of development has been previously defined and researched for the learning objective of computational thinking.

Prior studies in the fields of physics, biology, chemistry, and computer science in post-secondary education have demonstrated significant differences in the cognitive representations of learners at the extremes of the skill acquisition scale: novices versus experts. These differences have been taken as proxy indicators of skill development in the experts. This study proposes a theoretical framework combining the personal construct theory of Kelly (1955) with the Dreyfus model of skill acquisition (1987; 1980) to explain these differences as cognitive changes in an individual's organization and processing of skill knowledge which affect their recognition of and responses to stimuli resulting from experience gained through repeated practice of the skill. This theoretical framework is able to describe the intervening levels of computational thinking skill acquisition which may be observed in computer science students as they progress through their degree programs.

Therefore, by eliciting data from individuals regarding their current methods of organizing and processing skill related knowledge, an assessment may be drawn

regarding the progress of their development of skill competency. Repeated single-criterion card sorting activities effectively visualize individuals' cognitive processes for selecting criteria that both aggregate and differentiate among conceptual terms. Furthermore, objective, quantitative measures have been validated for the analysis of data collected from card sort instruments. The premise of this study is that card sorting activities offer a credible, valid, and objective means to quantitatively assess students' development of computational thinking skill.

Therefore, this study proposed to utilize a card sorting instrument from earlier research of higher education computer science students along with subsequently developed and validated quantitative data analysis methods to establish a baseline of structural exemplar sorts as potential markers of student development of programming related knowledge as proxies for measures of computational thinking ability relative to experience gained by practicing the skill. An investigative survey was conducted in the Spring semester of 2019 to administer a questionnaire and card sort activity to undergraduate students enrolled in computer science courses at the University. The data was analyzed to address the research questions suggested by the application of the theoretical framework to this data collection method.

A baseline of indicators of cognitive development of computational thinking ability resulting from this study can serve as an additional basis for formative assessments of student progress toward this learning objective. These assessments will enable adaptive instructional strategies for the benefit of students and continuous improvement of the curriculum for the benefit of the institution. Most importantly, results

of this study can direct future discussion and research into the evaluation of the benefits claimed for computational thinking.

CHAPTER IV

Results

Introduction

Card sorting activities have previously been used to assess the development of conceptual representations related to field-specific problem-solving skills in higher education students (Bissonnette, et al., 2017; Irby, et al., 2016; Krieter et al., 2016; McCauley et al., 2005). Historically, the data elicited through card sorting instruments did not lend itself to quantitative methods for analysis (Sanders, et al., 2005). That began to change in 2005 with Deibel et al.'s definition of the edit distance metric for comparison between pairs of card sorts. This new measure allowed the different studies cited above to employ a variety of quantitative methods, based on edit distances, to compare and contrast differences among the card sorts of independent samples of participants. This current study combines a number of these methods to analyze the collected data and answer its research questions.

Research Questions

The following research questions were addressed in this study:

1. Is there a relationship between categories of coursework achievement (Introductory, Mid-Program, Completing) and categories (High, Low, or Zero) of card sort orthogonality?
2. How many reliable and interpretable structural exemplar sorts are derivable from the collections of sorts produced by those participants most likely to have attained the desired level of programing skill?

3. Is there a statistically significant difference among the categories of computer science students' progression through milestones of coursework attainment (Introductory, Completing, and Mid-program) on the dependent variable, the edit distance between the card sorts to the exemplar sorts?
4. Is there a statistically significant difference among the categories of computer science students' programming experience (Light, Moderate, Extensive) on the dependent variable, the edit distance between the card sorts to the exemplar sorts?

Null Hypotheses

- H₀₁*: There is no statistically significant relationship between categories of coursework achievement (Introductory, Mid-Program, Completing) and categories (High, Low, or Zero) of card sort orthogonality.
- H₀₂*: No reliable and interpretable structural exemplar sorts are derivable from the collections of sorts produced by those participants most likely to have attained the desired level of programming skill.
- H₀₃*: There are no statistically significant differences among the categories of computer science students' progression through milestones of coursework attainment (Introductory, Completing, and Mid-program) on the dependent variable, the edit distance between the card sorts to the exemplar sorts.
- H₀₄*: There are no statistically significant differences among the categories of computer science students' programming experience (Light, Moderate, Extensive) on the dependent variable, the edit distance between the card sorts to the exemplar sorts.

Data Collection Results

Data collection occurred during the Spring 2019 semester. In February, a majority of computer science students were solicited for participation in the study by the instructors of the courses listed in Table 1 (see Chapter 3). Following approval from the IRB (see Appendix O), in April a second solicitation sought participation from the students in all sections of the introductory computer science course, COSC 1436. All instructors were encouraged to award extra credit as an incentive for students to participate in the study activities but their decision was not reported back to the study.

Data collection from the initial solicitation was conducted from February 20th through March 25th. Course instructors collected 109 signed forms for both informed consent and an equal number of forms for FERPA consent for release of academic records. During this collection period, a total of 84 students entered the online study website and completed the registration process and the survey instrument. Of these 84 participants, 75 also submitted one or more card sorts. Two participants subsequently withdrew from the study, resulting in usable surveys from 82 students and 195 card sorts from 74 students.

Data collection from the solicitation of students in the introductory computer science course, COSC 1436, was conducted from April 22nd through May 1st of 2019. Instructors for the course collected signed forms of informed consent from 64 students. Since this course is typically the first and only course of record in computer science for students enrolled in COSC 1436, there was no intent to collect their academic records, and so no forms for FERPA consent for release of academic records were distributed to nor collected from these students. During this second collection period, a total of 53

students entered the online study website and completed the registration process and the survey instrument. One or more card sorts were contributed from 50 of these participants, resulting in an additional 101 card sorts.

For this study, the objective was to collect data from three sources: a self-administered online survey, a self-administered online card sort activity, and official academic records provided by the Registrar's office. In total, the data collection for this study gathered usable surveys from 135 participants, with 124 of these providing 296 card sorts. Academic records were requested for 108 students enrolled in computer science courses beyond COSC 1436.

Data Derivations and Transformations

The online study website stored and formatted data as submitted by the participants in a manner designed for processing efficiency and accuracy. However, in order to analyze the results, it was necessary to both transform the structure of the collected data and to derive several additional measures from the data. This section details the procedures involved in these processes.

Participant Anonymization. In order to correctly associate each participant with their submitted data, the online system captured and authenticated the school email address of each participant every time they entered the website. In order to provide anonymity of the participants during data analysis, each student email address was assigned to a unique, randomly generated participant identifier during the initial online study registration process. All subsequent data collected online was coded with this participant identifier. Student email addresses were omitted from all data extractions, with only the participant identifier used to differentiate the submissions among

participants. Similarly, for each participant who consented to the release of their academic records, their email address was associated with a randomly generated academic identifier which was used by the Registrar's office to report the academic records to the study. Once these records were received by the study, they were recoded from the academic identifier to the participant identifier through their known association with the student email address.

Demographic categorizations. A procedural batch processing program was written in Node.js to reformat the survey data into a form more conducive for review and analysis. The results of this process were stored in a new table called participantTags. Each column in this table corresponds to a survey question. Since some of the online survey questions allowed for an open response, the transformation program assigned the most similar and frequent responses to a fixed categorical value. Ethnicity for example, included open responses such as: Asian, Asian-American, and South Asian. These were categorized as Asian in the participantTags table. Appendix H details the transformations that were performed on each survey field.

Other categorizations. Several other participant characteristics were categorized by ranking results obtained from survey responses, academic records, and card sort contributions. The survey asked for binary responses to twelve types of possible instruction in computer programming outside of SHSU coursework. A score for Prior Instruction was calculated by counting all true responses to these twelve questions. This score was then ranked against those of all participants. Two breakpoints were then identified which divided the rank list into three nearly equal-sized groups. Participants in the group with the lowest total scores were categorized as Light. Participants in the group

with the highest total scores were categorized as Extensive, and the remaining participants were categorized as Moderate.

The survey also asked for Likert style responses to four types of purposeful programming experience outside of SHSU coursework and scored on a scale from zero to four. Different weights were assigned for each question. A score for Purposeful Experience was calculated by multiplying the score for each question by the weight of that question, and then summing the results of each of these questions together. This score was then ranked against those of all participants. Two breakpoints were then identified which divided the rank list into three nearly equal-sized groups. Participants in the group with the lowest total scores were categorized as Light. Participants in the group with the highest total scores were categorized as Extensive, and the remaining participants were categorized as Moderate.

From the academic records, a cumulative grade point average (GPA) was calculated for all courses taken by the participant offered by the Department of Computer Science. This cumulative GPA was then ranked against those of all non-Introductory participants. Two breakpoints were then identified which divided the rank list into three nearly equal-sized groups. Participants in the group with the lowest cumulative GPA were categorized as Bottom. Participants in the group with the highest cumulative GPA were categorized as Top, and the remaining participants, including those introductory students in COSC 1436 were categorized as Average.

Also from the academic records, participants were assigned to one of five levels of coursework achievement: COSC 1436 denotes level A; either COSC 1437, COSC 2327, or DFSC 1316 is level B; either COSC 2329, COSC 3318, or COSC 3327 is level

C, and level D is denoted by a grade of D or better in either COSC 3319, COSC 4318, or COSC 4319. Level E is attainment of Level D along with more than 28 hours of COSC or DFSC courses and a grade point average of 2.0 or better for all COSC and DFSC coursework. These milestones were then categorized into three tiers: Introductory for Level A students; Mid-program for Levels B and C; and Completing for Levels D and E.

As described below, participants who contributed more than one card sort had a measure of orthogonality calculated for them. Participants who submitted only a single card sort were assigned an orthogonality measure of zero. This orthogonality value was then ranked against those of all participants who contributed card sorts. Two breakpoints were then identified which divided the rank list into three groups. Since 45% of cases were single-sorts, the lowest breakpoint was set just above an orthogonality measure Normalized Minimum Spanning Tree (NMST) of zero to create the Zero category. The second breakpoint was set after an NMST value of 7.0 which placed nearly equal percentages (27% and 28% respectively) of the frequency counts for the range of non-zero NMST values between 2 and the maximum of 11.4 for the remaining participants in the Low and High categories.

Derivation of edit distance between card sorts. This study chose to analyze collected card sort data by using two derived measures: the edit distance between pairs of sorts, and the orthogonality of a collection of sorts. The edit distance measure is defined by Deibel et al. (2005) as the number of items in one card sort which have to be moved between groups in order to exactly replicate the other card sort. Deibel et al. (2005) described the calculation of edit distance by utilizing an algorithm from bipartite graph theory known as the Hungarian, or Munkres algorithm for solving the Assignment

Problem. For this study, a procedural batch processing program was written that used a publicly available implementation of the Munkres algorithm for Node.js. The validity of this derivation procedure was verified using example card sorts and results published in Fossum and Haller (2005). For this study, the edit distance was calculated between each pair of the 296 submitted sorts. The result for each pair was stored in a table called editDistance. For the 296 sorts, this resulted in 43,660 unique edit distance pairs $[(296^2 - 296) / 2]$. See Appendix K for a listing of the program code, and the program execution.

Orthogonality among card sorts. The measure of orthogonality of a collection of sorts (NMST) is an indication of the degree of variance among the sorts. A low value indicates higher similarity while a high value denotes greater differentiation. An NMST value was derived for each participant and then ranked against those of the other participants as previously described.

Calculation of orthogonality requires the availability of edit distances between each pair of sorts in the collection to define a weighted, undirected graph where weights between graph vertices are the edit distances between the sorts represented by the vertices. Card sort orthogonality is defined by Fossum and Haller (2005) and is based on the concept of the minimum spanning tree in graph theory. Orthogonality is calculated as the length of a sort collection's minimum spanning tree divided by the total number of sorts in the collection (Fossum & Haller, 2005). For this study, a procedural batch processing program was written to use a publicly available Node.js implementation of the Eager Prim algorithm for determination of the minimum spanning tree of a weighted, undirected graph. The validity of this derivation procedure was verified using example card sorts and results published in Fossum and Haller (2005). Orthogonality results were

derived both for manually selected collections of sorts as well as for all of the sorts contributed by individual participants. An initial version of the orthogonality program derived the orthogonality measure for every participant who contributed one or more sorts and stored the results in a table called `participantStats`. This table also captured the total number of sorts in the collection, the identifier of each sort in the collection, the edit distances among each pair of these sorts, the path of the minimum spanning tree and its total length, and the mean, standard deviation, and range of the edit distances for the collection. A second version of the orthogonality program derived the same table data for researcher selected collections involving sorts from multiple participants and stored the results in a table called `collectionStats`. See Appendix L for a listing of the program code, and execution for each individual participant.

For participant-based analyses, the `participantTags` and `participantStats` tables are joined together in an SQL view to provide a dataset containing data collected from both survey and card sort instruments for every participant. This dataset was extracted from the database into an Excel spreadsheet (using a data connection), and then imported into SPSS and saved as a `.sav` file. See Appendix I for a listing of the SPSS variable data definition of the `.sav` file.

Descriptive participant statistics. SPSS version 25 (IBM Corp, 2017) was used for analysis of datasets. Descriptive statistics and frequency counts on participant gender, ethnicity, age, classification level, coursework milestone achievement level and category, intended major, prior instruction, purposeful experience, grade point average and GPA category, sort count, sort orthogonality, and self-assessments of experience with completion of programming assignments and competence as a programmer were

generated for the 135 participants. See Appendix J for the detailed frequency counts and statistics. The demographics of this sample were 23.7% female and 1.5% identifying as non-binary; 16.3% Asian, 17% Black, 20% Hispanic, 41.5% White and 5.2% categorized as other; ages ranged from 19 to 53 with 80% younger than 26. In terms of institutional classification level, 14.8% self-reported as Freshmen, 21.5% as Sophomores, 22.2% as Juniors, and 41.5% as Seniors; while based upon coursework milestone categorization as described above, 38.5% were considered Introductory level students, 23% were at the Completing level, with the remaining 38.5% classified as Mid-program.

In terms of the five milestone levels, 20% were at Level E, 28.1% were at Level C, 10.4% were at Level B, and 3% had attained Level D but lacked either the hours or the GPA requirements to be classified as Level E. Computer Science majors accounted for 45.9% of the participants, Software Engineering Technology were 16.3%, Digital Forensics were 6.7%, with 28.9% majoring in programs in Department of Technology Engineering, mostly Electronics and Computer Engineering Technology (ECET).

Participants were classified in the bottom third, Light, for prior instruction if they reported less than two non-SHSU training examples, and in the top third, Extensive, with more than three examples. Participants were classified in the top third, Extensive, for purposeful experience if their responses scored above 3.4 and in the bottom third, Light, if they reported no purposeful experiences.

Card sorts were submitted by 124 of the participants with 45.2% contributing only a single sort and 28.2% responding to the online prompts to contribute at least four sorts. Two participants produced 6 and 8 sorts respectively. The sort orthogonality of the top

third of respondents measured 7.0 or above, while the bottom third contributed only a single sort (an implicit orthogonality of zero).

The majority of participants (60%) reported that they struggled to complete programming assignments for their coursework with 41.5% requiring more than three hours per assignment, 14.1% having to seek help in order to complete their code, and 4.4% admitting that they *just don't get it* when it comes to programming. A third of participants (34.1%) reported that they typically completed assignments in less than three hours, and the remaining 5.9% reported that they could complete their programming assignments in less than two hours. When asked to self-assess their programming competency, 5.2% consider themselves to be *really good* and often sought out by others for help; 16.3% reported that they enjoy programming and experience few problems; 37% feel confident in their ability to complete their programming assignments; 34.8% consider themselves to be *beginner* programmers; and 6.7% admit that they *just can't code*.

RQ1 - Analysis of Relationship between Orthogonality and Categories of Achievement

To address the first research question, card sort results were cross-tabulated by categories of coursework achievement (Introductory, Mid-program, and Completing) and categories of participant NMST values. NMST values were categorized according to the distribution of frequency counts. Since 45 percent of the 124 participants who completed the card sort activity contributed only a single card sort, and therefore had an NMST value of zero, these single sort cases were labeled as the Zero category. The NMST values for the remaining participants were categorized as either Low or High in a manner

to place nearly equal percentages of frequency counts (27% and 28% respectively) in each category. The Low category thus contained NMST values ranging from 2.0 to 7.0, while the High category ranged from 7.19 to 11.4. A bar chart of the frequency counts from the cross tabulation is shown in Figure 9. Further details of the statistical analysis for this question may be found in Appendix C.

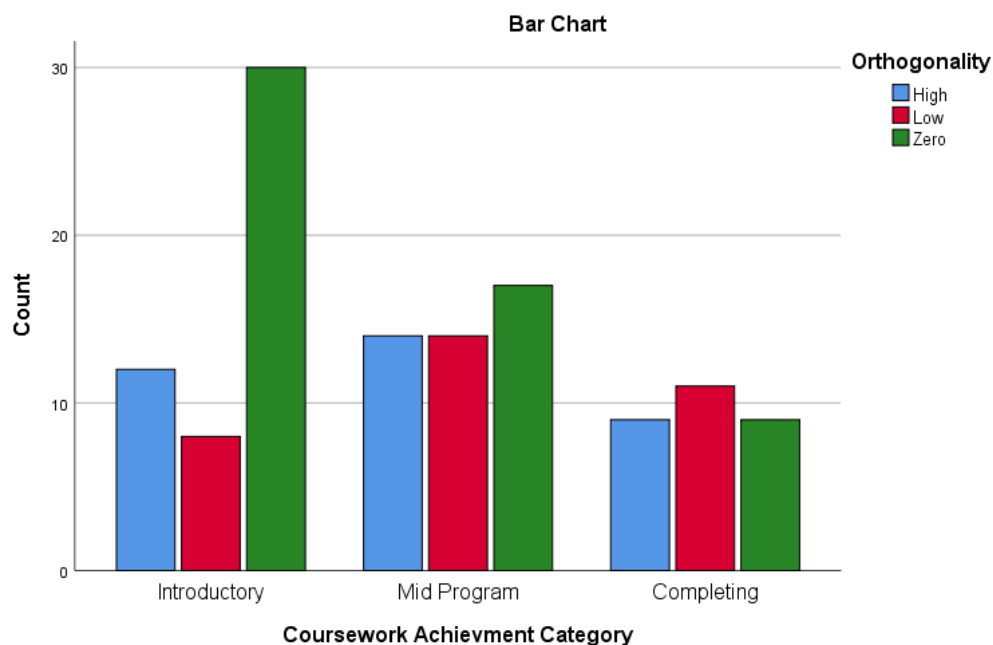


Figure 9. Bar chart of frequency counts from cross-tabulation of card sorts by coursework achievement category and category of orthogonality.

Examination of the counts and percentages indicated that the frequency of zero NMST values, representing single sort submissions, decreased as students progressed in their coursework, with 60% of Introductory students submitting single sorts, while 37.8% of Mid-program students, and 31% of Completing students submitted single sorts. Comparable numbers of participants submitted multiple card sorts among the three categories of students: 20 each for Introductory and Completing students and 28 for Mid-program students. NMST values were calculated for these participants and categorized as

either high or low. Mid program students had an equal percentage of high and low NMST values (31.1%). High NMST values, denoting greater differentiation among submitted card sorts, outnumbered low NMST values among the Introductory students (24% vs 16%), while Completing students had fewer high than low NMST values (31% high vs 37.9% low).

A chi-square was calculated to determine the statistical relationship between categories of coursework achievement (Introductory, Mid-program, and Completing) and categories of card sort orthogonality (Zero, Low, High). All cells of the cross-tabulation had frequency counts of five or more. There was no statistically significant relationship ($X^2(4) = 8.65, p > .05$) found between categories of coursework achievement and categories of card sort orthogonality. The effect size determined by Cramer's V was .19 indicating that a small association was found between student's orthogonality of their card sorts and whether the student was at the Introductory, Mid-program, or Completing milestone in their computer science coursework.

Summary of findings for research question 1. A statistical analysis of the relationship between computer science students' categories of coursework achievement and the orthogonality of their card sorts was conducted with a bivariate chi-square. No statistically significant evidence was found to reject the null hypothesis. The observed trend for this study was that a greater percentage of Introductory students sorts had high (24%) as opposed to low (16%) orthogonality values, while Completing students had a greater percentage of low (38%) to high (31%) orthogonality values. These results appear contrary to the findings of Fossum and Haller's (2005) comparison of NMST values

between introductory and senior computer science students in the datasets from Sanders et al. (2005) and McCauley et al. (2005).

RQ 2 - Identification of Card Sorts that Exemplify the Desired Level of Conceptual Development

Bissonnette et al. (2017) and Krieter et al. (2016) previously concluded that card sorts produced by students in the early stages of developing field specific problem-solving skills (e.g., for biology and chemistry, respectively) typically use surface factor sorting criteria, while those of faculty and graduate students typically use conceptual deep factor criteria. In card sort studies of computer science students, McCauley et al. (2005) performed a qualitative content analysis of 291 card sorts for 65 graduating seniors and classified the sorts into 16 Content Analysis Groups (CAGs), based upon the participants' descriptions of sorting criteria and category label. They calculated the orthogonality of the sorts within each of these CAGs and demonstrated that CAGs organized around a coherent theme were composed of similarly structured sorts which were confirmed by low NMST values. In essence, a quantitative procedure was performed to validate a qualitative analysis. This current study elected to follow a suggestion by Fossum and Haller (2005) to reverse the order of these procedures and first perform a quantitative clique analysis to identify groups of structurally similar sorts for subsequent content analysis.

The objective for research question 2 was to identify from the 296 submitted card sorts, a few structural exemplar sorts that represent deep factor conceptualizations of those participants most likely to have attained the desired level of programming skill. A

multiple step process was required to achieve this objective. This process is illustrated in Figure 10.

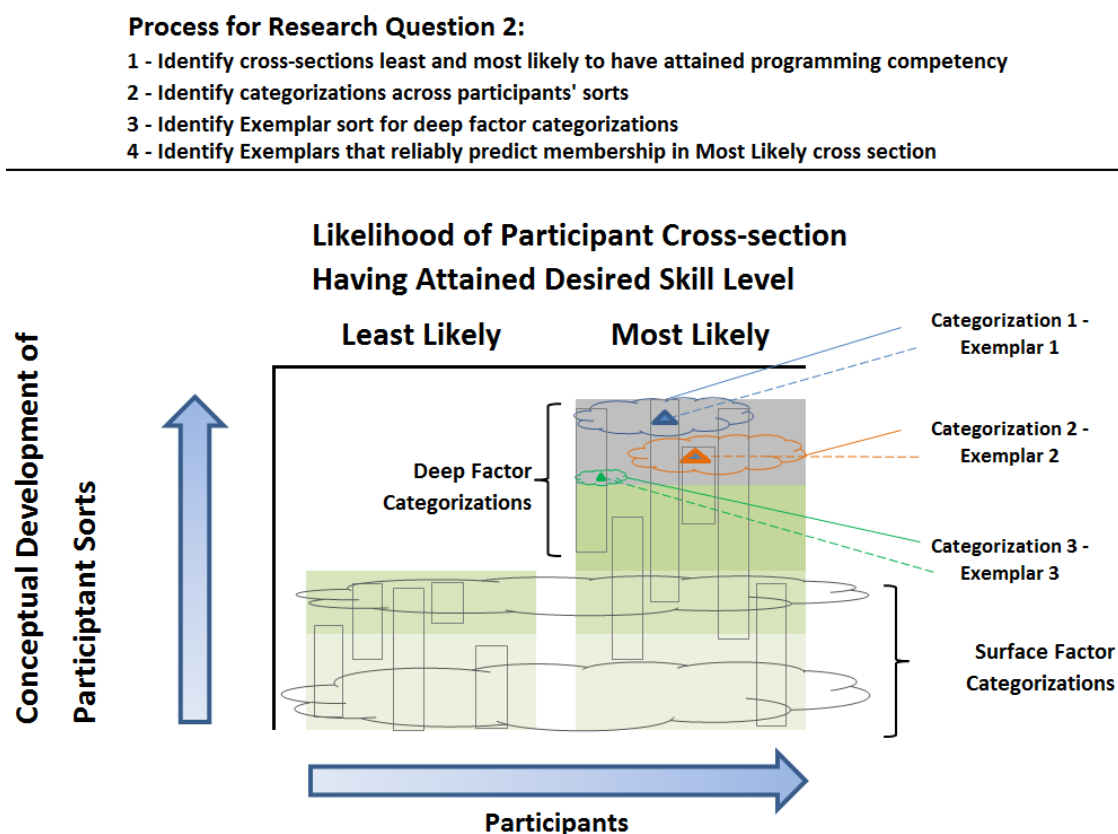


Figure 10. Illustration of multiple step process used to identify exemplar sorts of deep factor categorization for research question 2.

Process. As described in Chapter 3, participants were solicited from varied cross-sections of the population of students studying computer science at the University. These participants each completed a questionnaire that collected data such as their university classification level, intended major, and self-assessments of their ability to complete programming assignments and their overall competency as a programmer. These data were collected to provide a basis for partitioning the participants into cross-sections for comparison and analysis. An interactive tool was developed to allow the researcher to select participants according to these characteristics, as shown in Figure 11. Each

selection of participants defined a specific cross-section, which was referred to as a collection and identified with a unique collection number. The study database tables could then be queried to extract the statistics and card sorts for participants belonging to a specific collection by the collection number.

Computational Thinking Skills Study				Data Reviewer			
Collection 13 of 14							
Ethnicity: <input type="checkbox"/> Asian <input type="checkbox"/> Black <input type="checkbox"/> Hispanic/Latino <input type="checkbox"/> White <input type="checkbox"/> Other		Classification Level: <input type="checkbox"/> Freshman <input type="checkbox"/> Sophomore <input checked="" type="checkbox"/> Junior <input type="checkbox"/> Senior		Intended Major: <input type="checkbox"/> Computer Sci <input type="checkbox"/> SW Engnr <input type="checkbox"/> Digitl Forensics <input type="checkbox"/> non-CS <input type="checkbox"/> Other		Assignments: <input checked="" type="checkbox"/> < 2 hrs <input checked="" type="checkbox"/> < 3 hrs <input type="checkbox"/> > 3 hrs <input type="checkbox"/> Need Help <input type="checkbox"/> Don't Get It	
Gender: <input type="checkbox"/> Female <input type="checkbox"/> Male <input type="checkbox"/> non-Binary/Declined		Purposeful Exp: <input type="checkbox"/> Minimal <input type="checkbox"/> Average <input type="checkbox"/> Extensive		Prior Training: <input type="checkbox"/> Minimal <input type="checkbox"/> Average <input type="checkbox"/> Extensive		Performance: <input type="checkbox"/> Top <input type="checkbox"/> Average <input type="checkbox"/> Bottom	
Competence: <input checked="" type="checkbox"/> Really Good <input checked="" type="checkbox"/> Enjoy <input checked="" type="checkbox"/> Confident <input type="checkbox"/> Beginner <input type="checkbox"/> Can't Code		Achievement: <input type="checkbox"/> Introductory <input type="checkbox"/> Mid program <input type="checkbox"/> Completing					
Population Size: 124	Sample Size: 29	Sort Count: 76	NMST: 7.61842	ED Mean: 14.8239	Std Dev: 2.95887	Range: 0 to 22	
Description:				Action: <input type="checkbox"/> Revise the Criteria <input type="checkbox"/> Save the Collection <input type="checkbox"/> Cancel the Revision <input type="checkbox"/> View Cluster Analysis			

Figure 11. Participant characteristics for collection 13.

Fourteen collections in all (see complete list in Appendix N) were selected and investigated with an objective of identifying one collection presumed to represent the cross-section of participants most likely to have attained the desired level of programming skill, and another collection representing the cross-section least likely to have attained the desired skill level. The following collections were considered:

- Collection 3: Students classified by the University, as freshmen and sophomore in the Introductory course (55 sorts from 32 participants).
- Collection 6: Students classified by the University, as juniors and seniors who self-reported as normally completing programming assignments in three hours or less, with a GPA in the Average and Top categories (77 sorts from 27 participants).
- Collection 11: Students classified as seniors with a GPA in the Top (71 sorts from 20 participants). This collection includes participants who may have

self-reported having greater difficulty in completing programming assignments or having a lack of confidence in their programming abilities.

- Collection 13: Students classified as juniors and seniors who self-reported as normally completing programming assignments in three hours or less, and who are also self-confident in their programming abilities (76 sorts from 29 participants).

Collection 3, consisting of first- and second-year students in the Introductory course, was presumed to represent the cross-section least likely to have attained the desired skill level. Collections 6, 11, and 13 were potentially representative of the group most likely to have attained the desired level of programming skill. A cross-reference was constructed to show which of the 296 total submitted sorts had membership in these three collections. This activity identified sort sensitivity to the various selection characteristics of the collections. Additionally, the interactive tool shown in Figure 11 also provided the ability to visually recreate sorts within a collection. Using this ability, sorts that appeared in the cross-reference across more than one of the three collections (6, 11, or 13) were then visually recreated, as shown in Figure 12, and evaluated for their lexical content. This yielded approximately two dozen card sorts that were identified as potentially representative of participants' attainment of the desired level of programming skill. Although each of the collections contained some of these card sorts, the majority of them could be found within collection 13. Therefore, collection 13 was presumed to be the cross-section representative of participants most likely to have attained the desired skill level.

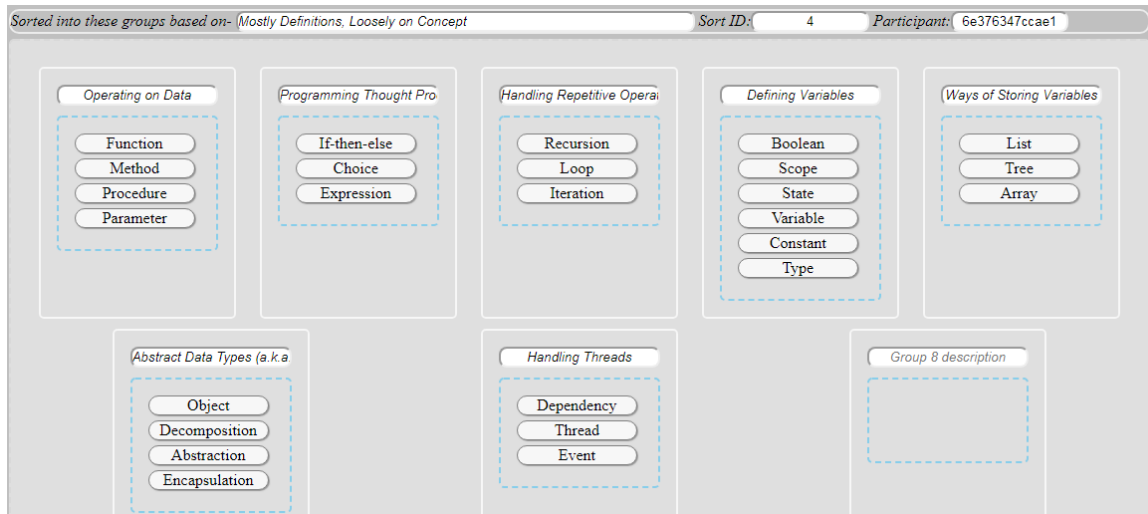


Figure 12. Display of sort Id 4 from collection 13.

The next step in the process of identifying exemplar sorts that represent deep factor conceptualizations was to categorize the sorts of the most-likely cross-section, collection 13, into coherent themes. Rather than perform a qualitative content analysis of the participant supplied sort criterion and group labels for all 76 card sorts in the collection, a quantitative data mining technique for identify cliques within a community of data points was utilized. Themes for each clique were then derived using the ability of the tool shown in Figure 11 to display the card sorts within each clique.

The analysis of cliques in data mining derives from graph theory. A clique is a subset within a community of vertices in which all members of the clique have weighted edges to all the other members of the clique (i.e., the graph is complete) and all of these edges are less than or equal to a specified length, d (Deibel et al., 2005; Johnston, 1976). Cliques occur as a result of variances in the distribution of edit distances among the card sorts. Selection of the d value is obviously a significant factor in the identification and meaningfulness of cliques. Cliques with a d value of zero contain card sorts that are structurally identical, i.e., where all elements have been grouped together in an identical

number of groups. Increasing the d value permits more card sorts to be associated with the clique, with the consequence that the sorts of the clique overall become more dissimilar (i.e., the NMST value of the clique increases). Eventually the d value reaches a point where the conceptual theme that represents the clique is too broad to be meaningful. Therefore, the researcher should evaluate cliques within a considered range of d values (Deibel et al., 2005).

This current study followed an iterative approach in searching for cliques within a specified range of d values. For each collection of participants, statistics were calculated for the range of edit distances among all sorts in the collection, and the mean and standard deviation of these edit distances. For the clique analysis of that collection, a lower d -value was then set equal to the bottom of the range of edit distances in the collection. The upper d -value was set as the lesser of the value eight, or the mean edit distance less one standard deviation. Restricting the upper d -value to the mean less one standard deviation limits the number of eligible sort pair edges in a collection to a maximum of 16%, thereby promoting cliques that are more cohesive and distinct within the community. Likewise, imposing a maximum upper value of eight was selected as that edit distance represents one third of the theoretical maximum distance between two card sorts of 26 items, so any greater edit distances are increasingly indicative of too great a dissimilarity.

A Node.js program was written to identify cliques within a collection by iterating within this specified range to remove all edges of length $> d$ from the edit distance table of sorts within the selected collection (Deibel et al., 2005). Each iteration resulted in a graph where not all vertices have an edge to all other sorts (i.e., an incomplete graph). A

publicly available implementation of the Bron-Kerbosch algorithm from graph theory was then utilized to identify all cliques within this remaining graph (Johnston, 1976). The identified cliques were recorded in a database table along with the current d -value, the list of member sorts, the edges among the sorts, the minimum spanning tree path, and the NMST value for the clique. The interactive tool shown in Figure 11 displayed the analysis results for a collection, as shown in Figure 13.

Computational Thinking Skills Study						Data Reviewer					
Cluster Analysis for			Collection- seq: 13	ID: 13	sort Count: 76	nmst: 7.61842	ED Mean: 14.8239	Range: 0	to: 22		
seq	nmst	d-Size	sort Cut	Select	sort Ids	nmst Path					Instance
91	3.5	8	2	●	["157","21"]	(21,157);7;					15
92	3.5	8	2	●	["185","186"]	(185,186);7;					20
93	3.66667	7	3	●	["91","92","93"]	(91,93);4; (91,92);7;					22
94	3.66667	8	3	●	["219","218","220"]	(218,219);4; (219,220);7;					11
95	3.66667	8	3	●	["91","93","92"]	(91,93);4; (91,92);7;					30
96	3.75	8	4	●	["5","21","74","74"]	(4,5);5; (4,74);5; (21,74);5;					24
97	4	7	4	●	["6","206","155","201"]	(6,206);5; (201,206);6; (155,201);5;					17
98	4	8	3	●	["130","155","131"]	(130,131);4; (130,155);8;					4
99	4	8	2	●	["131","219"]	(131,219);8;					7
100	4	8	2	●	["131","269"]	(131,269);8;					8

Figure 13. Clique analysis results for collection 13 showing d values of 7 and 8.

Cliques within collection 13 were examined to understand which cliques at smaller d sizes gained additional sorts as the d size of the analysis was increased. This process highlighted the significance of several cliques at the largest d size. Based upon a content comparison of the sorts contained within these cliques, cohesive themes for the categorization represented by the cliques were identified. These themes aligned with categories of sorts, such as CAGs, identified in earlier card sort studies of computer science students (Deibel et al., 2005; McCauley et al., 2005). Examples included *What I know / Don't know* (sorts 6, 58, 130, 194, 206), *Concrete / Abstract* (sorts 139, 201), and *Types of Programming Terms* (sorts 4, 15, 21, 51, and 74).

For each clique theme, a procedure defined by Deibel et al. (2005) was followed to identify the single sort within the clique that is most equidistant to all other sorts in the clique (see Appendix M). This designates the structural exemplar for that clique. Each exemplar represented a potential deep factor categorization and provided a suitable basis

for comparing the similarity of participant card sorts against that deep factor. At the conclusion of the above process, eight structural exemplars from collection 13 were identified: sorts 6, 15, 21, 84, 85, 185, 193, and 201.

These eight structural exemplars characterize the range of sorts frequently submitted by participants selected in collection 13, the juniors and seniors most comfortable with their programming skills. The next step was to determine whether these sorts adequately differentiate between the submissions of participants representing the highest concentration of students with the desired level of conceptual development for programming (collection 13) from the submissions of students representing the least likely to have attained the desired level of conceptual development (collection 3: freshmen and sophomores in the introductory course). This evaluation was performed by calculating edit distances between the card sorts of participants in collections 3 and 13, and for each participant, recording the shortest edit distance, i.e., the proximity, to each of the eight structural exemplar sorts. Also recorded was whether the participant is a member of the desired collection 13 group. Note that a lower proximity value indicates that participant's sort is more similar to the exemplar, while a higher proximity value indicates less similarity to the exemplar.

Analysis. Proximities to these eight identified exemplars were initially tested as indicators of significant difference between participants in collection 3 versus collection 13 using independent samples *t*-tests. An examination of the proximity values in preparation for conducting the *t*-tests identified three outliers for proximity to exemplar 6. These cases were removed from the analysis.

Data values within the groups for each *t*-test were checked for normal distribution and homogeneity of variance. Kolmogorov-Smirnov statistics and z-scores for skewness and kurtosis were calculated to evaluate the frequency distributions. Z-scores for all of the groups fell within the range of ± 3 . Groups from collection 13 had Kolmogorov-Smirnov significance values above .05 indicative of normal distribution, while the collection 3 groups had values below .05. However, the collection 3 group sizes each exceeded 20. Based on the z-scores and group sizes, normal distribution of the proximity values was assumed for both groups.

Levene statistics were calculated for the proximities to each exemplar to evaluate homogeneity of variance. Equal variances were found for the proximities to exemplars 84, 85, 185 and 201. Unequal variances were found for the proximities to exemplars 6, 15, 21, 193. Results of the *t*-tests were interpreted accordingly (Cohen, Manion, & Morrison, 2011).

Statistically significant differences between participants in collection 3 and collection 13 (see Appendix D) were found for the mean proximities to exemplars 6 [$t(28.2) = 3.09, p = .004$], 21 [$t(31.1) = 2.47, p = .019$], 185 [$t(53) = 2.29, p = .026$], and 201 [$t(53) = 2.23, p = .03$] respectively. Proximities to exemplars 15, 84, 85, and 193 in collections 3 and 13 were not found to be statistically significantly different ($p > .05$). Therefore, those exemplars were removed from further consideration as differentiators.

To determine which of the independent variables (proximities to exemplars: 6, 21, 185, and 201) are reliable predictors of computer science students' attainment of student outcome 2 (least likely, most likely) based upon level of development of conceptual representations of programming concepts, forward binary logistic regression was

performed. Data screening led to the elimination of 4 cases of multivariate outliers. Predictors were tested in groups of 3 in order to keep the ratio of cases to predictors above 15: 1 (Pallant, 2001). Regression results (see Appendix D) indicated that the overall model of two predictors (proximity to exemplar 6 and proximity to exemplar 21) was statistically reliable in distinguishing between sorts submitted by members of collection 13 (most likely to have attained the desired skill level) and members of collection 3 (least likely to have attained the desired skill level) [-2 Log likelihood = 51.76, $\chi^2(2) = 19.63$, $p < .001$]. The model correctly classified 15 of 23 (65.2%) as being members in collection 13, and correctly classified 73.1 % of cases overall. Regression coefficients are presented in Table 6.

Wald statistics indicate that the proximity of a student's sorts to both exemplar 6 and exemplar 21 significantly predicted their level of attainment of the desired level of programming skill. The odds ratios for these two variables indicate a decrease in the likelihood of having attained the learning objective as the proximity value to either of these exemplars increases. Stated conversely, a closer proximity to the exemplars increases the likelihood of a student demonstrating a level of conceptual development representative of having attained the desired level of programming skill for student outcome 2.

Table 6

Logistic Regression Coefficients for Proximity to Exemplars 6 and 21

	<i>B</i>	<i>Wald</i>	<i>df</i>	<i>p</i>	Odds Ratio
Exemplar 21	-.358	6.57	1	.010	.699
Exemplar 6	-.328	8.40	1	.004	.720
Constant	9.259	9.89	1	.002	

Summary of findings for research question 2. The objective for research question 2 was to identify structural exemplar sorts that best represent the deep factor conceptualizations of those participants most likely to have attained the desired level of programming skill. The card sorts of participants considered to be most likely to have attained deep factor conceptualizations were quantitatively analyzed to identify cliques of highly similar sorts. Based upon a content comparison of contained sorts, cliques were selected that best represented a cohesive criterion for categorization and which aligned with previously identified categories of sorts, such as Content Analysis Groups (McCauley et al., 2005). Selected cliques were then mathematically reduced to structural exemplar sorts (Deibel et al., 2005). Using standard statistical analysis procedures, the measures of participants' sorts proximity to the exemplar sorts 6, 21, 185, and 201 were found to yield statistically significant differences between the groups of participants considered the *least*, and the *most* likely to have attained the desired level of programming skill. Two of these measures, proximities to exemplar 6 and to exemplar 21, were found to be reliable predictors of participants' membership in the *most likely* group.

While quantitative means and measures were utilized to identify these structural exemplars, it is worth examining the qualitative considerations represented by these four submitted card sorts before addressing the subsequent research questions. These exemplars are relatable to the CAGs categorized by McCauley et al. (2005). Using the ability of the interactive analysis tool mentioned above, each sort has been visually recreated as shown below.

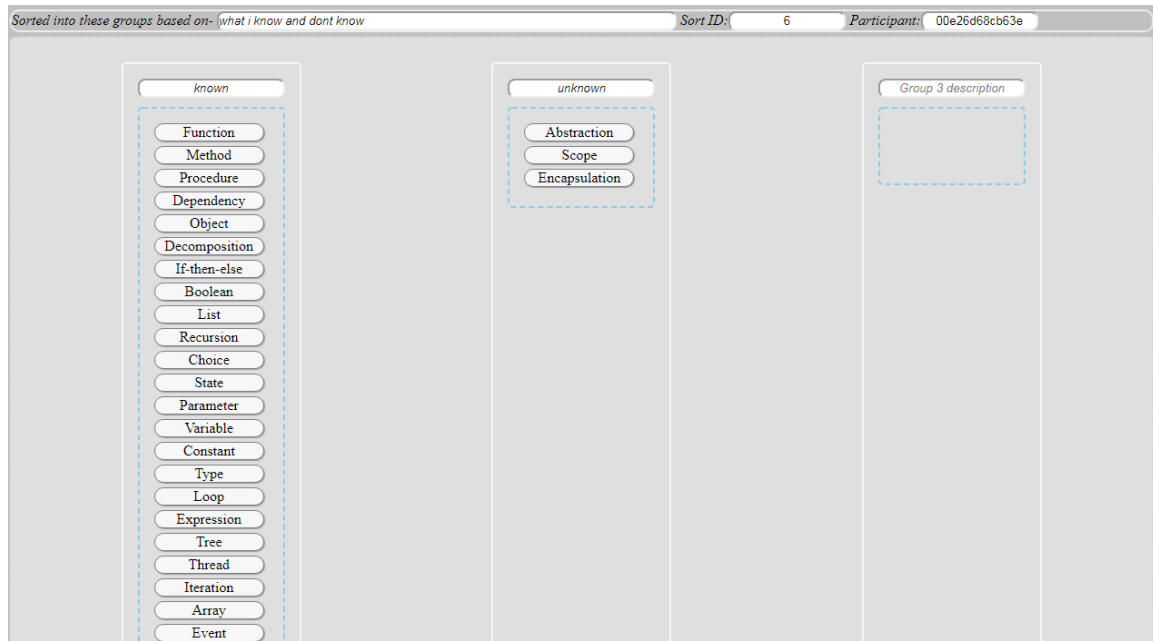


Figure 14. Exemplar 6, representing the *Know / Don't Know* criterion.

Exemplar sort 6 (Figure 14) is of the CAG category *What I know / Don't know*. Examples of this criterion were widely found among the submissions of participants in this study. However, the grouping of elements between the two categories was observed to differ significantly between the introductory students, and the completing students. Therefore, Exemplar 6 serves as an example of the fewer number of programming terms which should remain in the unknown group upon completion of coursework for a computer science degree. Introductory students may be expected to have many more elements in the unknown group and thus to have a higher edit distance from this exemplar.

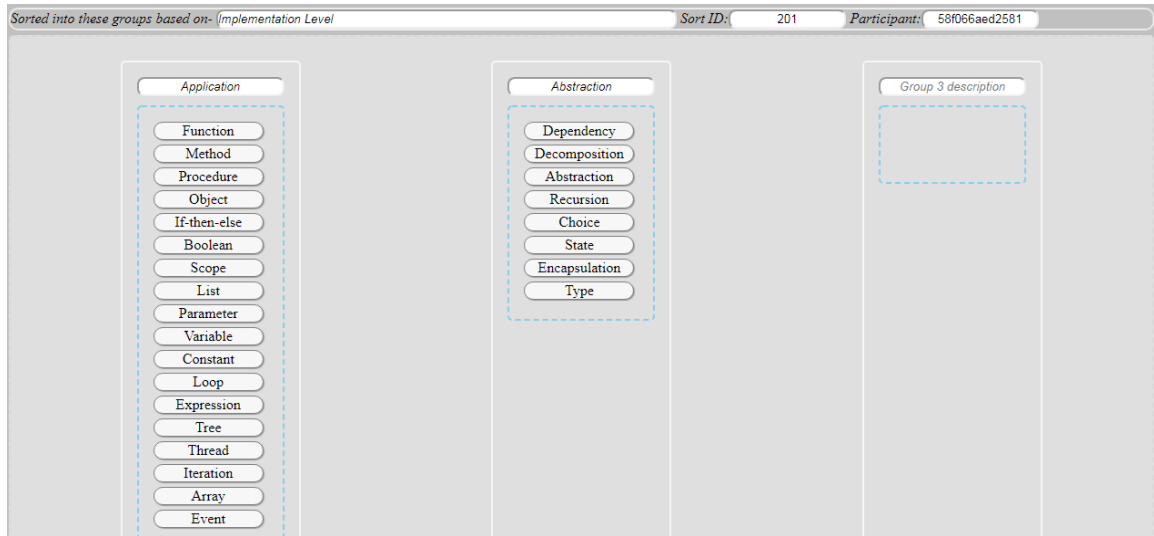


Figure 15. Exemplar 201, from the *Concrete / Abstract* CAG.

Exemplar 201 (Figure 15) is an example of the *Concrete / Abstract* CAG.

Although it is dichotomous like exemplar 6, this criterion is based on a deeper conceptual representation which might be expected to develop later in students' coursework. Close proximity values to this exemplar seem to represent this more conceptual, or deep factor dichotomy.

Exemplars 21 and 185 are both examples of the *Types of Programming Terms* CAG. Students using this criterion group terms together based upon their conceptual representations of fundamental programming concepts. The better developed the representations, the more differentiated the groups will be. Examples of this criterion were submitted by many participants, using widely varying numbers of groups. Exemplar 185 (Figure 16) may represent an intermediate level of conceptualization, where relationships between terms may be clearer in initial groups, but become less apparent in subsequent groups.

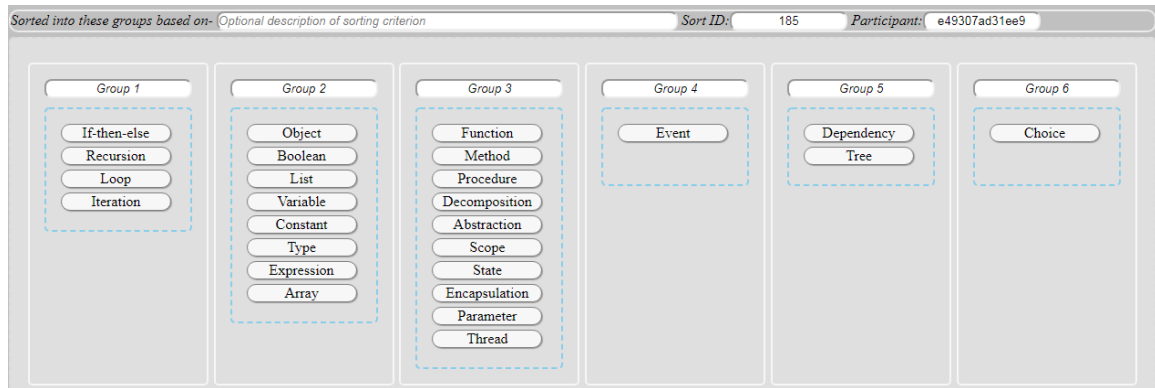


Figure 16. Exemplar 185, an intermediate *Types of Programming Terms* CAG example.

Exemplar 21 (Figure 17) presents groups that are clearly-defined and express an elaborate differentiation.

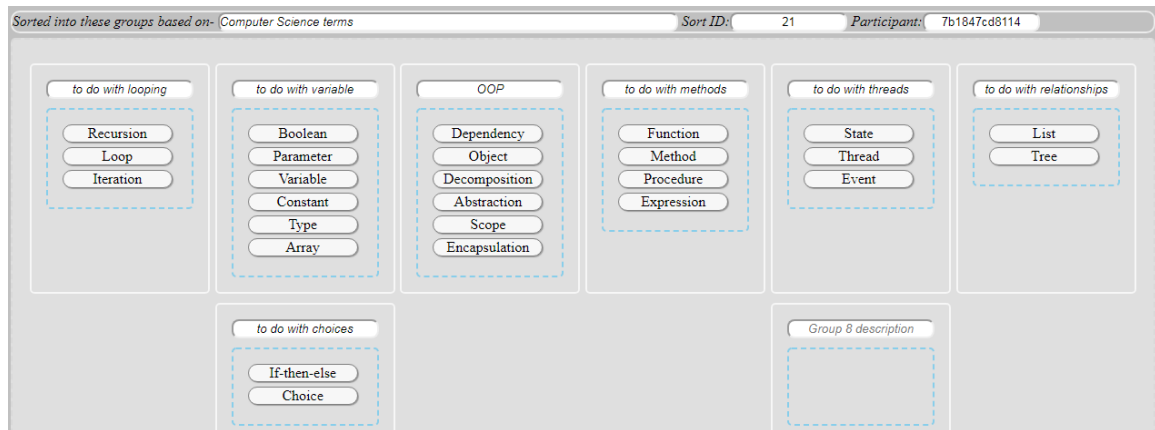


Figure 17. Exemplar 21, an advanced *Types of Programming Terms* CAG example.

Finally, two zero cliques (d value = 0) were identified in many of the collections, including both collection 3 and 13. Zero cliques are significant because they consist of card sorts that are identical. This high degree of similarity among card sorts submitted by different participants was achievable because these sorts categorized on the basis of surface factors readily apparent to participants, such as the first letter of each programming term, and the number of letters in each term. Examples are shown in Figure 18.

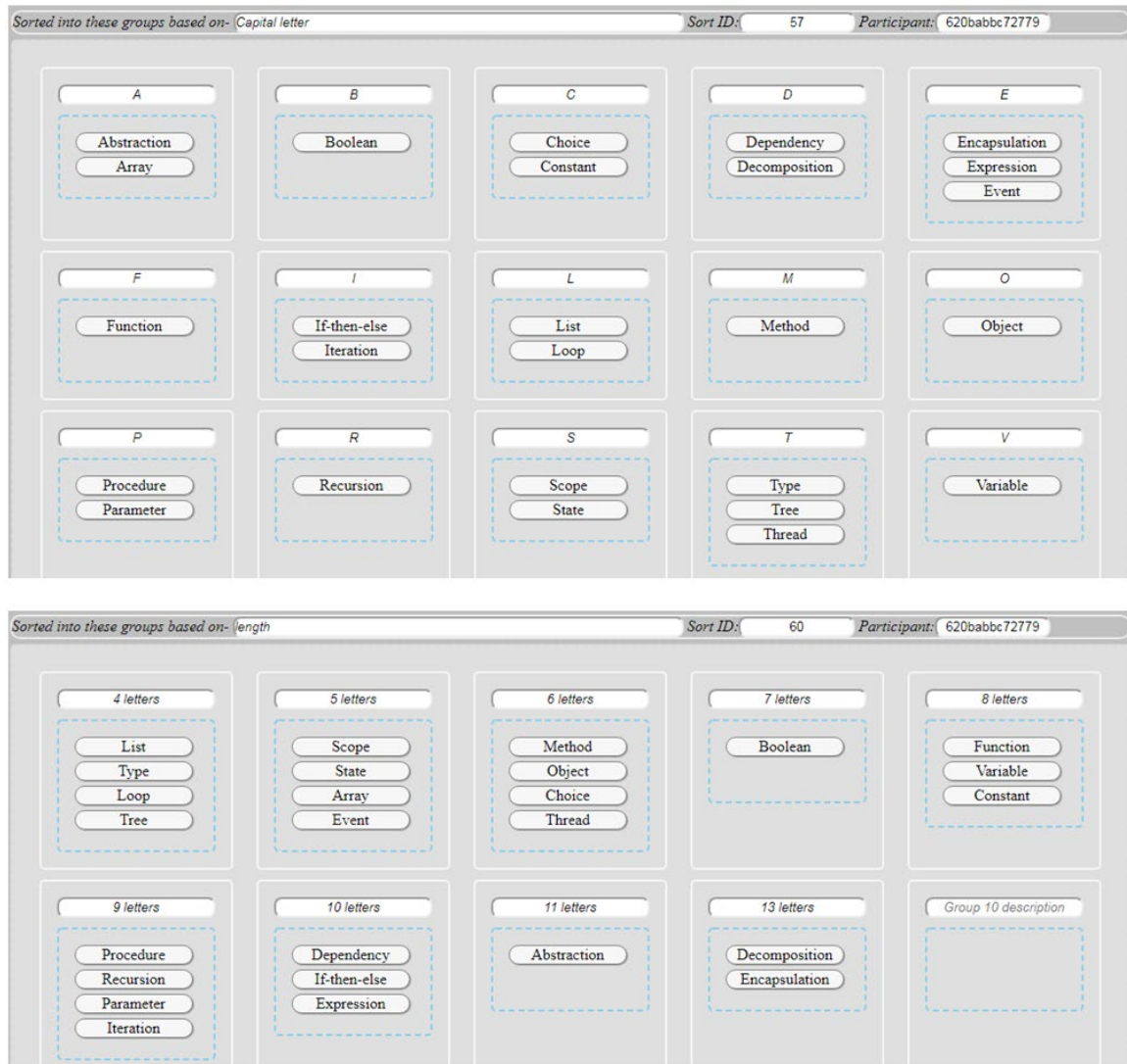


Figure 18. Surface factor sort grouping alphabetically (top) and by length (bottom).

RQ3 - Analysis of Variances between Proximity to Exemplar Sorts by Coursework Attainment

Given the four exemplar sorts identified for the previous question, Research Question 3 examines whether the proximal distance between computer science students' card sorts and each of these exemplar sorts significantly differs according to participant's progression through milestones of coursework attainment (introductory, mid-program, and completing). A node.js program was written to identify for each participant the

shortest edit distance between each exemplar sort (sorts 6, 21, 185, and 201) and every sort submitted by that participant. This resulted in an edit distance to each of the four exemplars for each participant. These edit distances were recorded in the participantTags table, as that participant's proximity values to the four exemplars. The participantTags table was then imported into SPSS for an analysis of the variance of mean proximity values to an exemplar sort in a one-way ANOVA as grouped by the three categories of coursework attainment. A separate ANOVA was conducted for each exemplar sort and the details of these analyses are reported in Appendix E.

Initial examination of the proximity values in preparation for conducting the ANOVAs identified a few outliers that fell below two standard deviations from the group mean. These data were recoded into new variables to the lowest value within two standard deviations from the mean. This choice of value preserves recognition of the participant's close proximity to the exemplar without further skewing the distribution (Mertler & Vannatta, 2013).

Data values within the groups for each ANOVA were checked for normal distribution and homogeneity of variance. Kolmogorov-Smirnov statistics and z-scores for skewness and kurtosis were calculated to evaluate the frequency distributions. While the majority of the groups had Kolmogorov-Smirnov significance values lower than .05, the z-scores for all of the groups fell within the range of ± 3 . Additionally, each group size exceeded 25. Therefore, normal distribution of the proximity values was assumed for all groups.

Levene statistics were calculated for the proximities to each exemplar to evaluate homogeneity of variance. Equal variances were found for the proximities to exemplars

185 and 201, and Bonferroni was chosen for post hoc analysis. Unequal variances were found for the proximities to exemplars 6 and 21, and Dunnett T3 was selected for post hoc analysis as it has been shown to yield conservative results with unequal variances with equal or unequal group sizes (Shingala & Rajyaguru, 2015).

Box and whiskers plots of the groups analyzed by each ANOVA are presented in Figure 19. Note that higher proximity values indicate that a participant's nearest sort has low similarity with the exemplar; while lower proximity values indicate the participant sort is more similar to the exemplar. Examination of the mean proximities for each level of coursework attainment indicate that proximity to each of the four exemplars generally decreased, becoming more similar to the exemplar, relative to progressive levels of coursework attainment (introductory, mid-program, and completing).

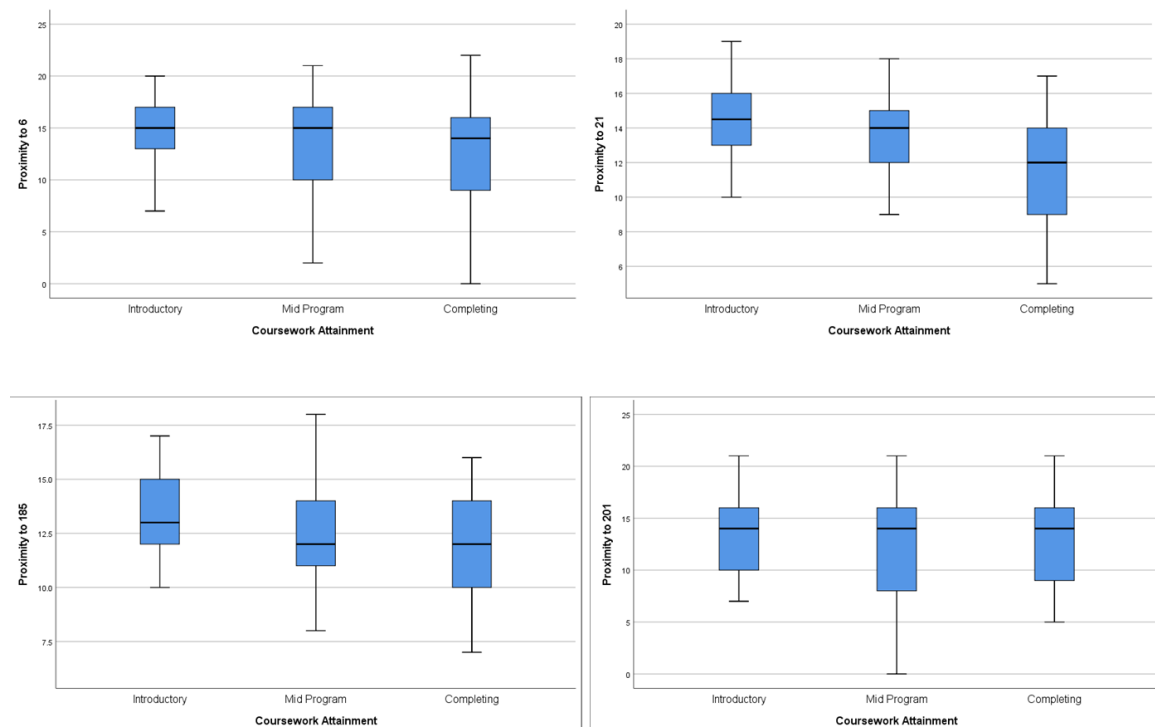


Figure 19. Box and whiskers plots of proximities to each exemplar sort by categories of coursework attainment.

One-way ANOVAs were conducted to compare categories of coursework attainment (Introductory, Mid-program, and Completing) with the proximity of participant sorts to each of the exemplar sorts. For exemplar sort 6 [$F(2,121) = 1.15, p > .05, \eta^2 = .02$], and exemplar sort 201 [$F(2,121) = .641, p > .05, \eta^2 = .01$], the differences among the means were not found to be statistically significant.

For sort proximities to exemplar sort 185, a statistically significant difference was found ($F(2,121) = 3.35, p = .038, \eta^2 = .05$). The effect size was $\eta^2 = .05$; a small effect size (Kirk, 1996) indicating that 5% of the variance of proximities to exemplar sort 185 was explained by a student's level of coursework attainment. Bonferroni post hoc tests found a statistically significant difference ($p = .038$) in the means of completing students ($m = 11.97, sd = 2.37$) to introductory students ($m = 13.24, sd = 2.04$).

A visual comparison of these means is presented in Figure 20. Introductory student sorts display the furthest proximity from, and the least similarity with, exemplar 185 while Completing student sorts display the nearest proximities and greatest similarity to the exemplar. The post hoc analysis found that the sorts of Completing computer science students are statistically significantly more similar to exemplar sort 185 than the sorts of the Introductory category of participants. While the Mid-program category of participants display a mean proximity to exemplar sort 185 which is between the means of the other two categories, participant sorts do not become significantly more similar to exemplar sort 185 until the Completing category of coursework has been attained.

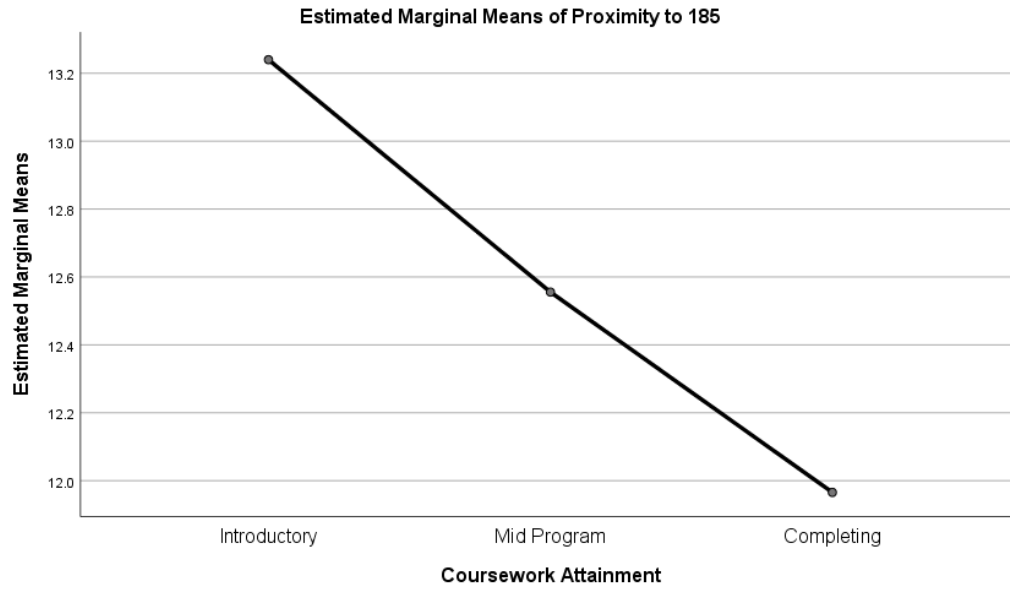


Figure 20. Comparison of group means for proximity to exemplar sort 185.

For sort proximities to exemplar sort 21, a statistically significant difference was found [$F(2,121) = 13.32, p < .001, \eta^2 = .18$]. The effect size was $\eta^2 = .18$; a large effect size (Kirk, 1996) indicating that 18% of the variance of proximities to exemplar sort 21 was explained by a student's level of coursework attainment. Post hoc tests using Dunnett T3 found a statistically significant difference between the means of Completing students ($m = 11.45, sd = 3.37$) to Mid-program students ($m = 13.58, sd = 2.29$) ($p = .013$), and to Introductory students ($m = 14.46, sd = 2.09$) ($p < .001$).

A visual comparison of these means is presented in Figure 21. Introductory student sorts display the furthest proximity from, and the least similarity with, exemplar 21 while Completing student sorts display the nearest proximities and greatest similarity to the exemplar. The post hoc analysis found that the sorts of Completing computer science students are statistically significantly more similar to exemplar sort 21 than the sorts of either the Introductory or Mid-program categories of participants.

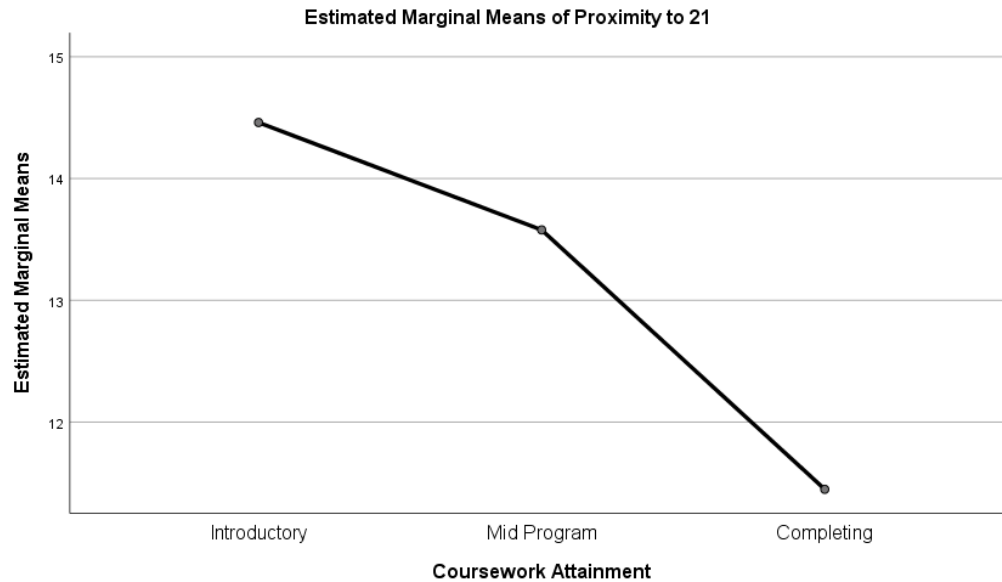


Figure 21. Comparison of group means for proximity to exemplar sort 21.

Summary of findings for research question 3. A statistical analysis of variances of the proximal distance between computer science students' card sorts to each of the exemplar sorts relative to participant's degree of coursework attainment (introductory, mid-program, and completing) was conducted using one-way ANOVAs. Statistically significant evidence was found to reject the null hypothesis in the use of exemplar sorts 21 and 185. Variances in distance between participants' card sorts to these two exemplars decreased, indicating a movement toward greater similarity, as students progressed in their level of coursework attainment from Introductory, to Mid-program, and to Completing. The direction of this trend in decreasing distance from the exemplars as coursework progresses appears consistent with the expectation of the theoretical framework that additional levels of instruction and study should yield indications of more advanced conceptual representations.

RQ 4 - Analysis of Variances Between Proximity to Exemplar Sorts by Programming Experience

Research Question 4 examines whether the proximal distance between computer science students' sorts and each of these exemplar sorts significantly differs according to participant's level of purposeful programming experience (light, moderate, and extensive). Using the proximity values for each participant to each exemplar sort as calculated for the previous research question, an analysis was made of the variance of mean proximity values as grouped by the three categories of programming experience to an exemplar sort using a one-way ANOVA. A separate ANOVA was conducted for each exemplar sort and the details of these analyses are reported in Appendix F.

Initial examination of the proximity values in preparation for conducting the ANOVAs identified a few outliers that fell below two standard deviations from the group mean. These data were recoded in new variables to the lowest value within two standard deviations from the mean. This choice of value preserves recognition of the participant's close proximity to the exemplar without further skewing the distribution (Mertler & Vannatta, 2013).

Data values within the groups for each ANOVA were checked for normal distribution and homogeneity of variance. Kolmogorov-Smirnov statistics and z-scores for skewness and kurtosis were calculated to evaluate the frequency distributions. While the majority of the groups had Kolmogorov-Smirnov significance values lower than .05, the z-scores for all of the groups fell within the range of +/- 3. Additionally, each group size exceeded 35. Therefore, normal distribution of the proximity values was assumed for all groups.

Levene statistics were calculated for the proximities to each exemplar to evaluate homogeneity of variance. Equal variances were found for the proximities to exemplars 21 and 185, and Bonferroni was chosen for post hoc analysis. Unequal variances were found for the proximities to exemplars 6 and 201, and Dunnett T3 was selected for post hoc analysis as it has been shown to yield conservative results with unequal variances with equal or unequal group sizes (Shingala & Rajyaguru, 2015).

Box and whiskers plots of the groups analyzed by each ANOVA are presented in Figure 22. Note that higher proximity values indicate that a participant's nearest sort has low similarity with the exemplar; while lower proximity values indicate the participant sort is more similar to the exemplar. Examination of the mean proximities for each level of programming experience indicate that proximity to each of the four exemplars generally increased, becoming less similar to the exemplar, between the light and moderate levels of programming experience. However, as programming experience increased from moderate to extensive, mean proximity values to exemplars 6 and 201 also decreased to levels nearer that of participants with light experience. For exemplars 21 and 185, mean proximity values of participants with extensive experience continued to increase relative to both light and moderate categories of experience.

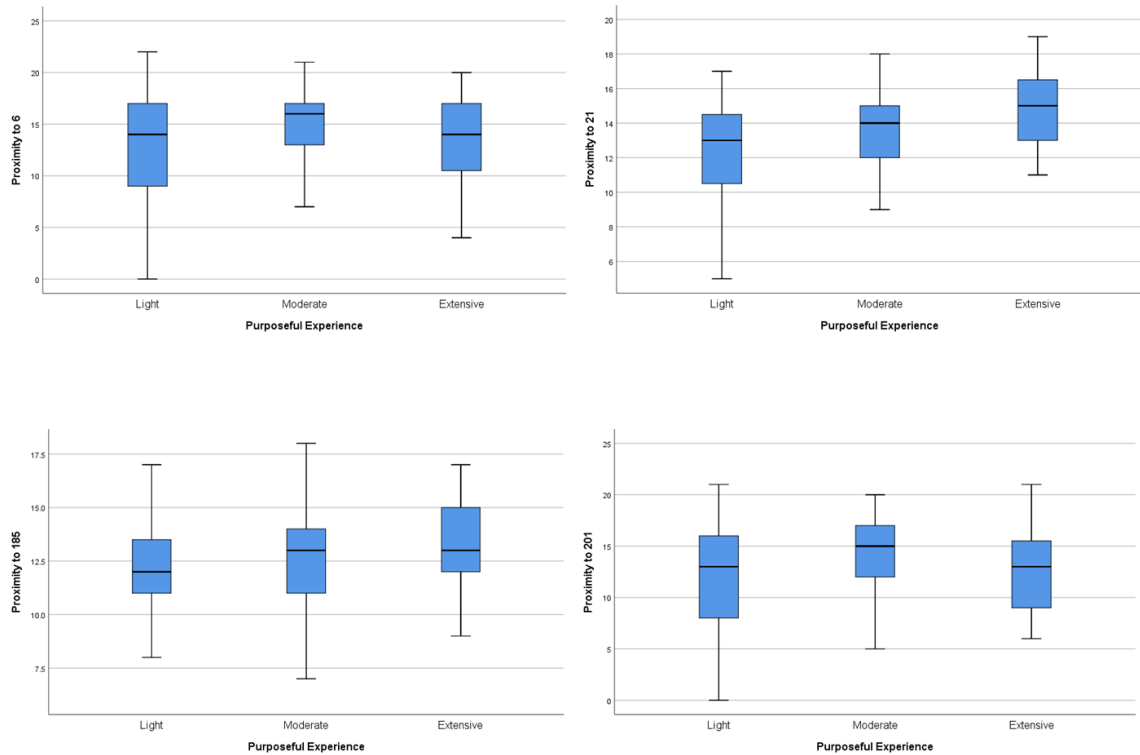


Figure 22. Box and whiskers plots of proximities to each exemplar sort by categories of purposeful programming experience.

One-way ANOVAs were conducted to compare categories of programming experience (Light, Moderate, and Extensive) with the proximity of participant sorts to each of the exemplar sorts. For exemplar sort 6 [$F(2,121) = 1.78, p > .05, \eta^2 = .03$], and exemplar sort 201 [$F(2,121) = 2.06, p > .05, \eta^2 = .03$] the differences among the means were not found to be statistically significant.

For sort proximities to exemplar sort 185, a statistically significant difference was found [$F(2,121) = 3.23, p = .043, \eta^2 = .05$]. The effect size was $\eta^2 = .05$; a small effect size (Kirk, 1996) indicating that 5% of the variance of proximities to exemplar sort 185 was explained by a participant's level of coursework attainment. Bonferroni post hoc tests found a statistically significant difference ($p = .039$) in the means of students with

extensive programming experience ($m = 13.33$, $sd = 2.0$) compared to students with light programming experience ($m = 12.16$, $sd = 2.13$).

A visual comparison of these means is presented in Figure 23. Examination of the mean proximities for each level of purposeful experience indicate that distances to the exemplar increased, indicating less similarity to the exemplar, relative to progressive levels of programming experience (light, moderate, and extensive). Post hoc analysis indicates that participants with light programming experience had sorts significantly more similar to the exemplar than students with extensive programming experience.

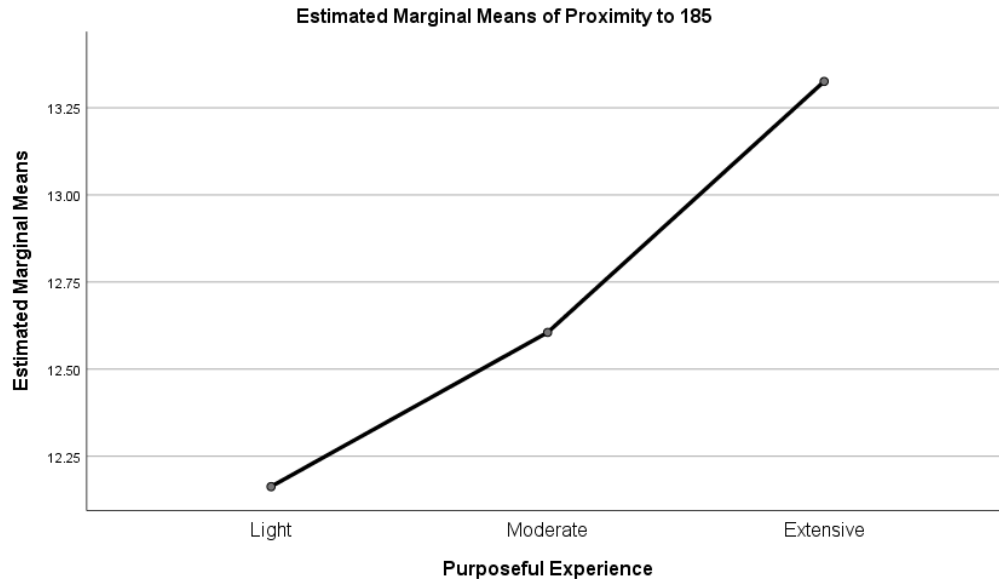


Figure 23. Comparison of group means for proximity to exemplar sort 185.

For sort proximities to exemplar sort 21, a statistically significant difference was found [$F(2,121) = 8.04$ $p = .001$, $\eta^2 = .12$]. The effect size was $\eta^2 = .12$; a medium to large effect size (Kirk, 1996) indicating that 12% of the variance of proximities to exemplar sort 21 was explained by a student's level of purposeful programming experience. Bonferroni post hoc tests found a statistically significant difference ($p < .001$)

in the means of students with extensive programming experience ($m = 14.58, sd = 2.13$) compared to students with light programming experience ($m = 12.37, sd = 3.12$).

A visual comparison of these means is presented in Figure 24. Participants with light programming experience had sorts with the nearest proximity to the exemplar while students with extensive programming experience had sorts with the furthest proximities to the exemplar. Post hoc analysis found that the sorts of computer science students with light programming experience are statistically significantly more similar to exemplar sort 21 than the sorts of participants with either moderate or extensive levels of programming experience.

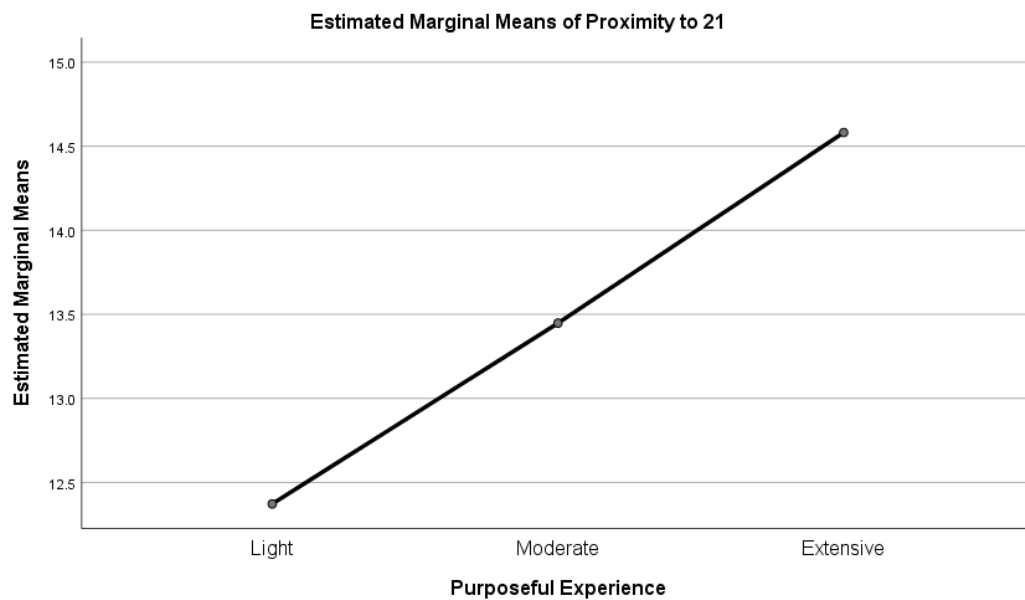


Figure 24. Comparison of group means for proximity to exemplar sort 21.

Summary of findings for research question 4. A statistical analysis of variances of the proximal distance between computer science students' card sorts to each of the exemplar sorts relative to participants' degree of programming experience (light, moderate, and extensive) was conducted using one-way ANOVAs. Statistically significant evidence was found to reject the null hypothesis in the use of exemplar sorts

21 and 185. Variances in distance between participants' card sorts to these two exemplars increased, indicating a movement toward less similarity, as students progressed in their degree of programming experience from light, to moderate, and to extensive. The direction of this trend in increasing distance from the exemplars as experience increases, appears contrary to the expectation of the theoretical framework that increases in experience should yield indications of more advanced conceptual representations.

Summary of Findings

This study tested four Null Hypotheses and obtained the following results:

H₀₁: There is no statistically significant relationship between categories of coursework achievement (introductory, mid-program, completing) and categories (high, low, or zero) of card sort orthogonality.

No statistically significant evidence was found to reject the null hypothesis.

The trend observed in this study was that a greater percentage of introductory students' card sorts had high (24%) as opposed to low (16%) orthogonality values, while completing students had a greater percentage of low (38%) to high (31%) orthogonality values. These results appear contrary to the findings of Fossum and Haller's (2005) comparison of NMST values between introductory and senior computer science students.

H₀₂: No reliable and interpretable structural exemplar sorts are derivable from the collections of sorts produced by those participants most likely to have attained the desired level of programing skill.

Four exemplar sorts were isolated and found to yield statistically significant differences between the groups of participants considered the *least*, and the

most likely to have attained the desired level of programming skill. Two of these variables, proximities to exemplar 6 and to exemplar 21, were found to be reliable predictors of participants' membership in the *most likely* group to have attained the learning objective.

H₀₃: There are no statistically significant differences among the categories of computer science students' progression through milestones of coursework attainment (introductory, completing, and mid-program) on the dependent variable, the edit distance between the card sorts to the exemplar sorts.

Statistically significant evidence was found to reject the null hypothesis in the use of exemplar sorts 21 and 185. Variances in distance between participants' card sorts to these two exemplars decreased, indicating a movement toward greater similarity, as the level of coursework attainment progressed from introductory, to mid-program, and to completing. The direction of this trend in decreasing distance from the exemplars as coursework progresses appears consistent with the expectation of the theoretical framework that additional levels of instruction and study should yield indications of more advanced conceptual representations.

H₀₄: There are no statistically significant differences among the categories of computer science students' programming experience (light, moderate, extensive) on the dependent variable, the edit distance between the card sorts to the exemplar sorts.

Statistically significant evidence was found to reject the null hypothesis in the use of exemplar sorts 21 and 185. Variances in distance between participants'

card sorts to these two exemplars increased, indicating a movement toward less similarity, as the degree of programming experience increased from light, to moderate, and to extensive. The direction of this trend in increasing distance from the exemplars as experience increases, appears contrary to the expectation of the theoretical framework that increases in experience should yield indications of more advanced conceptual representations.

Further discussion of these results and additional investigation into their meaning is presented in the following chapter, Discussion and Conclusion.

CHAPTER V

Discussion and Conclusion

Introduction

The area of study for this dissertation was Computer Science education. Specifically, the study focused on an objective for developing students' competency in programming at the post-secondary level according to accreditation criterion for student outcomes. Student Outcome 2 of the Accreditation Board for Engineering and Technology (ABET, 2019) states:

Graduates of the program will have an ability to . . . design, implement, and evaluate a computer-based solution to meet a given set of computing requirements in the context of the program's discipline. (Criterion 3. Student Outcomes).

The abilities listed in Student Outcome 2 require development of competency in the myriad analytical and problem-solving skills, often referred to as Computational Thinking skills (Wing, 2006), required for effective program design and development. In reflecting upon research into the evaluation of program effectiveness in computer science education since 2006, Denning (2017) summarized the problem facing computer science educators: after a decade of research and academic discussion about the definition and assessment of computational thinking, "we have no consensus on what constitutes the skill and our current assessment methods are unreliable indicators" (p. 36). He called for the development of baseline measures of programming expertise levels modeled after a framework of skill acquisition (Denning, 2017).

Review of literature found that a proxy predictor of a subject's problem-solving skill level can be obtained by eliciting the subject's basis for categorizing knowledge

through the use of a card sorting task (Mason & Singh, 2011). Recent studies in the fields of biology and chemistry at the post-secondary level have used card sort instruments to assess student development of conceptual development and related analytical and problem-solving skills (Bissonnette et al., 2017; Krieter et al., 2016). Searches for similar research in computer science education identified two gaps. First, there is an absence of studies in computer science education that are comparable to that for chemistry education to evaluate development of conceptual expertise. The Krieter et al. (2005) study used quantitative measures and methods to investigate the differences between the cognitive representations of introductory and senior students, and noted a trend toward development of expected knowledge representations as students progressed through the curriculum. The most recent studies to compare card sorts of introductory and senior computer science students were last conducted in 2005 when Fossum and Haller (2005) and McCauley et al. (2005) evaluated a unique measure of card sort orthogonality. The second gap is a lack of follow-up to the results of those prior studies in computer science which indicated that the conceptual representations of the lowest performing quartile of graduating seniors as measured by the orthogonality of their card sorts did not develop significantly beyond those of introductory students, while the top quartile of graduating seniors did demonstrate a statistically significant growth in their conceptual organization related to programming skills.

This study addressed these gaps in three steps. First, it replicated the previous studies of conceptualizations of computer science students to assess the use of the card sort orthogonality measure (NMST) as a differentiator between the top and bottom performing segments of senior students against introductory students. Second, it

categorized the card sorts and identified exemplars which differentiate between those students demonstrating significant conceptual growth from those students who are just beginning their conceptual development. Finally, it analyzed how progress toward the desired level of conceptual representation may be affected by students' levels of instruction and programming experience at various milestones through the curriculum.

This study used the knowledge elicitation instrument (a repeated, open card sort of 26 programming terms) and quantitative measures from McCauley et al. (2005) and Fossom and Haller (2005). Cross-sections of students in the computer science degree program at the University were solicited for the study with 124 participants contributing 296 card sorts. An analysis approach adapted from Krieter et al. (2016) and Bissonnette et al. (2017) compared differences in computational thinking skill acquisition as measured by card sort orthogonality, and proximal distances to the differentiating exemplars among cross-sections of the degree program partitioned by levels of achievement of coursework milestones and also by levels of programming experience.

Four Null Hypotheses were tested with the following results:

H₀₁: There is no statistically significant relationship between categories of coursework achievement (introductory, mid-program, completing) and categories (high, low, or zero) of card sort orthogonality.

No statistically significant evidence was found to reject the null hypothesis.

The trend observed in this study was that a greater percentage of introductory students' card sorts had high (24%) as opposed to low (16%) orthogonality values, while completing students had a greater percentage of low (38%) to high (31%) orthogonality values. These results appear contrary to the findings

of Fossum and Haller's (2005) comparison of NMST values between introductory and senior computer science students.

H₀₂: No reliable and interpretable structural exemplar sorts are derivable from the collections of sorts produced by those participants most likely to have attained the desired level of programing skill.

Four exemplar sorts were isolated and found to yield statistically significant differences between the groups of participants considered the *least*, and the *most* likely to have attained the desired level of programming skill. Two of these variables, proximities to exemplar sort 6 and to exemplar sort 21, were found to be reliable predictors of participants' membership in the *most likely* group to have attained the learning objective.

H₀₃: There are no statistically significant differences among the categories of computer science students' progression through milestones of coursework attainment (introductory, completing, and mid-program) on the dependent variable, the edit distance between the card sorts to the exemplar sorts.

Statistically significant evidence was found to reject the null hypothesis in the use of exemplar sorts 21 and 185. Variances in distance between participants' card sorts to these two exemplars decreased, indicating a movement toward greater similarity, as the level of coursework attainment progressed from introductory, to mid-program, and to completing. The direction of this trend in decreasing distance from the exemplars as coursework progresses appears consistent with the expectation of the theoretical framework that additional

levels of instruction and study should yield indications of more advanced conceptual representations.

H₀₄: There are no statistically significant differences among the categories of computer science students' programming experience (light, moderate, extensive) on the dependent variable, the edit distance between the card sorts to the exemplar sorts.

Statistically significant evidence was found to reject the null hypothesis in the use of exemplar sorts 21 and 185. Variances in distance between participants' card sorts to these two exemplars increased, indicating a movement toward less similarity, as the degree of programming experience increased from light, to moderate, and to extensive. The direction of this trend in increasing distance from the exemplars as experience increases, appears contrary to the expectation of the theoretical framework that increases in experience should yield indications of more advanced conceptual representations.

This chapter discusses the meaning and implications of the above results as they relate to existing literature, to the theoretical framework, and to the implications for teaching, learning, and assessing computer programming and computation thinking skills and abilities. It concludes with recommendations for further research and for establishing a baseline of quantitative measures for such assessment purposes.

Discussion

Using the NMST measure for differentiating conceptual development levels.

The card sorting activity is an effective instrument for eliciting a subject's basis for aggregating elements of field specific knowledge (Rugg & McGeorge, 2005).

Additionally, when the subject is prompted to repeatedly categorize the same stimuli, subjects with more differentiated conceptual representations tend to produce a larger number of card sorts with varied categorizations (Fossum & Haller, 2005). The NMST measure of structural dissimilarity (orthogonality) among a collection of card sorts produced by a single individual is one indication of the individual's capacity for categorizing a stimuli set using multiple, varied criteria. Therefore, the NMST measure has been proposed as a differentiator in assessing relative skill levels among computer science students (Fossum & Haller, 2005). Such a proposition was evaluated in two related studies of computer science students (McCauley et al., 2005; Fossum & Haller, 2005). This current study began its analysis of the collected card sorts by attempting to replicate the findings these prior studies. Each of these studies followed a repeated single-criterion sort technique and used the identical stimuli set of 26 programming terms.

McCauley et al. (2005) collected 291 card sorts from 65 graduating seniors at eight institutions of higher education in the United States. The grade point averages in computer science courses were also collected and used to partition the sample into performance quartiles. The NMST measure was calculated for each participant, and then the means for each quartile were compared. The results are shown in Table 7. The researchers found that the mean NMST increased with each step increase in quartile of GPA performance from the bottom to the top, and that the difference between the top and bottom mean values was statistically significant ($z = 2.97, p = .0025$) using a Mann-Kendall test for randomness against a monotone trend.

Table 7

NMST by Performance Quartile as Reported in the McCauley et al. (2005) Study

Quartile:	Bottom	Third	Second	Top	Total
N =	17	16	16	16	65
Sort count	69	75	62	86	291
Mean NMST	6.16	6.63	7.10	8.33	7.04

Note. Adapted from McCauley et al. (2005) Table 3

Fossum and Haller (2005) compared this data from the McCauley et al. (2005) study against a set of 1199 card sorts from 243 novice programming students and 33 experienced graduate students and faculty at 22 higher education institutions collected by Sanders et al. (2005). In initially comparing the mean NMST of all seniors against all novice students, the researchers were surprised to find a lack of difference. However, when the mean NMST of all but the bottom quartile of seniors (the senior+ group) was compared against the mean of all novice students, the difference was statistically significant ($z = 1.77, p < .04$, one-sided) using a Wilcoxon two-sample test (Fossum & Haller, 2005). Similar comparisons against the educators found a difference ($z = 1.67, p < .05$) with the full set of seniors, but no significant difference ($z = 0.66, p < .51$) with the senior+ group (Fossum & Haller, 2005). The researchers concluded from these findings, that either the NMST measure is inadequate as a differentiator among undergraduate students, or that the bottom quartile of seniors underperformed at meeting the expectations for graduating seniors.

As reported in Chapter 4, the current study examined the relationship between card sort orthogonality and three categories of coursework achievement (Introductory,

Mid-program, and Completing). No statistically significant association was found. The observed trend, as shown in Figure 25, was that the greater percentage of Introductory student sorts had high (24%) as opposed to low (16%) orthogonality values, while Completing students had a greater percentage of low (38%) to high (31%) orthogonality values. These results appear contrary to the findings in Fossum and Haller's (2005) comparison of NMST values between introductory and senior computer science students. Consequently, further analysis of the NMST values collected in this study was undertaken to understand these differences from the prior studies.

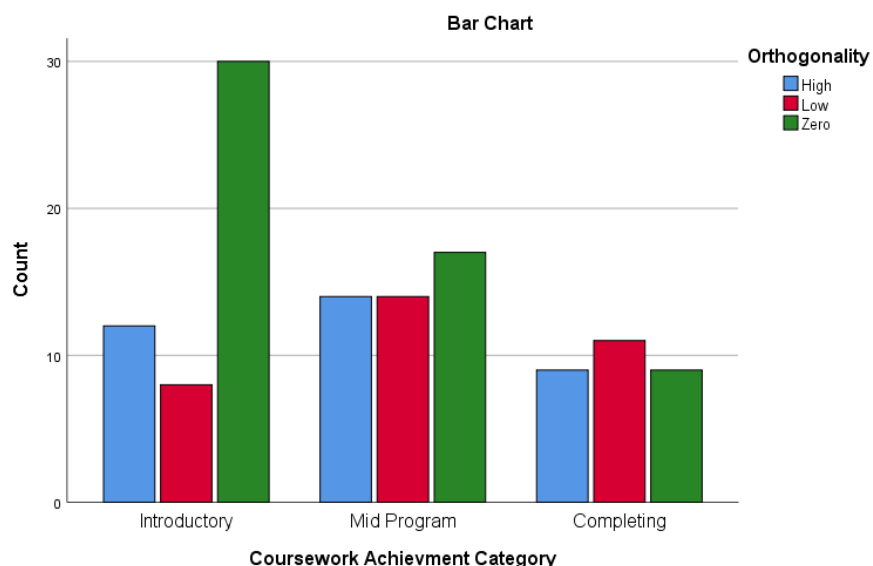


Figure 25. Bar chart of frequency counts from cross-tabulation of card sorts by coursework achievement category and category of orthogonality.

Also apparent from Figure 25 is the significant number of students who submitted only single-sorts and therefore had NMST values of zero. These cases skewed the means for each category of coursework achievement downward and by unequal amounts. Sixty percent of the contributions from introductory students were single card sorts, while 69% of completing students submitted multiple card sorts. McCauley et al. (2005) did not

report the number of graduating seniors who contributed only single card sorts. However, in Fossum and Haller's (2005) comparison of novices to seniors, it was noted that only 2 of 276 participants contributed single card sorts. Therefore, in order to perform a more comparable analysis, this study removed the single-card sort cases before performing the following tests.

The differences in mean NMST values were compared for all non-introductory participants who submitted at least two card sorts ($n = 48$) categorized into thirds (Top, Average, Bottom) according to cumulative computer science grade point averages. After determining that the three frequency distributions were normal and had equal variances, a one-way factorial analysis was conducted (see Appendix G) and found no statistically significant difference on the entire model [$F(2,45) = .20, p = .82, \eta^2 = .009$]. Although not statistically significant, the trend for mean values of NMST, as shown in Table 7 and Figure 26, increased with each step increase in GPA ranking category (from Bottom to Top) which is a finding consistent with McCauley et al. (2005).

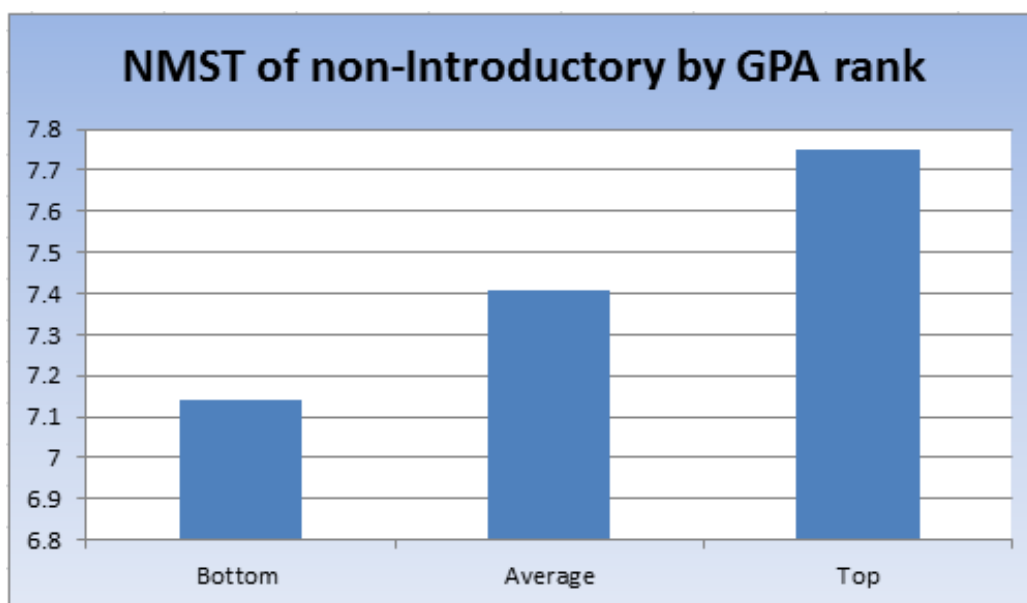


Figure 26. Bar chart of NMST means by category of GPA ranking ($n = 68$).

However, when the NMST mean for introductory students who contributed two or more sorts is added to the comparisons in Table 7, as shown in Table 8, it is apparent that its value is second only to the category of Top GPA students. This finding is contrary to results in Fossum and Haller (2005) which equated the mean NMST of novices to those of the Bottom GPA seniors. It is therefore surprising to find that the group of introductory students in this study had such a high mean NMST.

Table 8

*Mean NMST Values for Introductory Students Versus Non-Introductory Students
Categorized by GPA Ranking*

Category:	Introductory	Bottom GPA	Average GPA	Top GPA
N =	20	12	20	16
Mean NMST	7.62	7.14	7.41	7.75

Note. Adapted from McCauley et al. (2005) Table 3

To better understand how the NMST values of the introductory group differed from those of other cross-sections of participants, trends in the frequencies of NMST values were explored relative to other categories of collected data. As shown Figure 27, the means trended higher for both groups (introductory, non-introductory) when categorized by the variables programming experience and prior instruction. Participants categorized as Extensive in these variables had higher NMST values than those categorized as Light. Programming experience relates to types of self-reported programming activities outside of coursework assignments. Prior instruction relates to formal and informal training in programming received outside of the University degree program. It could be expected from the theoretical framework that participants reporting

higher levels of either of these variables would have more developed programming skills as reflected in higher NMST values.

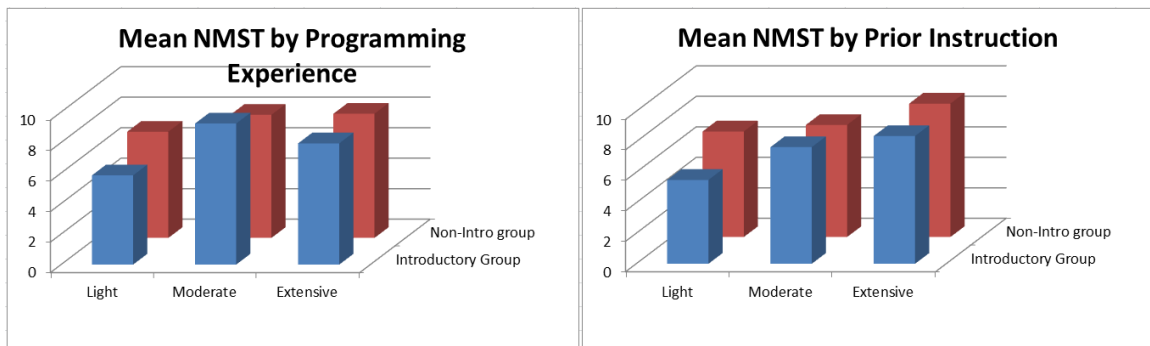


Figure 27. Comparison of mean NMST of introductory and non-introductory participants categorized by programming experience (left) and by prior instruction (right).

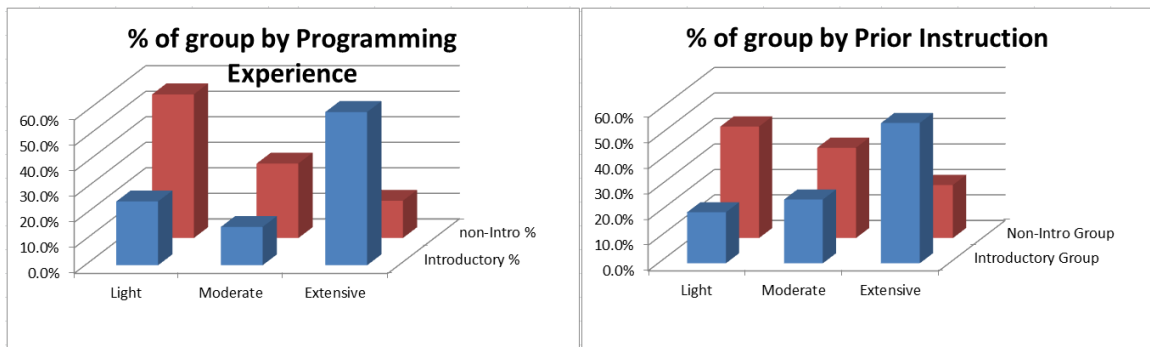


Figure 28. Distribution (by percentage) of participants among the categories for programming experience and prior instruction between the introductory and non-introductory groups

Figure 28 examines the distribution of participants, as percentages, between the Light and Extensive categories for the introductory and non-introductory groups. For both variables, the majority of introductory participants were categorized as Extensive. Taken in conjunction with Figure 27, this indicates that a majority of the introductory group had high NMST values, while only a minority of the group had low NMST values. This is consistent with the findings for research question 1 shown in Figure 25. Figure 28 also shows the situation reversed for the distribution for non-introductory participants with the majority categorized as Light, and therefore having the lowest NMST values.

This difference in instruction and experience outside of formal coursework between the introductory and non-introductory participants offers an explanation for why the mean NMST value of the introductory group was higher than expected relative to the categories of GPA ranking for the other participants as reported in Table 8.

Several conclusions regarding the use of mean NMST measures of card sort orthogonality as a differentiator of problem-solving skill levels of computer science students may be drawn from this study. First, cases of participants completing only a single card sort can skew mean orthogonality values lower, and variances in the percentage of such cases among groups may adversely affect statistical comparisons. Sanders et al. (2005) conducted 276 one-on-one facilitated card sorting activities using physical index cards, and collected only two single-sort cases. This study used an online, asynchronous tool to conduct 296 card sorts with 124 participants and collected 56 single-sort cases (45%). For statistical analyses using the NMST measure, the single-sort cases reduced the sample size from 124 to 68, which greatly reduced statistical power. While the online card sorting tool included prompts to encourage participants to complete multiple card sorts, this may have been less effective in this regard than the facilitated activities of the Sanders et al. (2005) study.

Second, for participants whose academic records were collected (the non-introductory group), the trend observed in McCauley et al. (2005) of mean NMST values increasing with categories reflecting increasing GPA performance was replicated. The card sorts of top performing participants were more differentiated than those of lower performing participants. However, in the broader comparison of introductory versus non-introductory participants, it became apparent that the NMST mean values were more

influenced by participants' exposure to programming experiences and prior instruction outside of their University coursework than by GPA ranking. Thus, the introductory group may have averaged higher NMST values than the bottom GPA ranking group due to a greater percentage of introductory students with greater exposure to experience building activities.

Therefore, based on the above, the NMST measure alone is not a sufficient basis for differentiating the level of conceptual development or problem-solving skill among computer science students.

Using exemplars of desired categorization for differentiating conceptual development levels. Card sorting instruments can be designed to elicit an individual's framework for categorizing field specific knowledge (Chi et al., 1981). These revealed frameworks can be effective in differentiating levels of conceptual development among individuals (Smith, 1990; Mason & Singh, 2011). Studies in the fields of physics, biology, and chemistry have demonstrated that putative experts tend to categorize based upon field specific concepts and principles implied in a stimuli set while novices tend to categorize using more readily apparent surface features of the stimuli (Chi et al., 1981; Bissonnette, et al., 2017; Krieter et al., 2016).

In card sort studies of the development of conceptual expertise of physics, biology, and chemistry students, researchers designed stimuli sets to elicit expected surface and deep factor categorizations (Bissonnette, et al., 2017; Krieter et al., 2016). Given the expected categorizations, researchers were able to construct exemplar sorts for each expected result. The similarity of participants' sorts to these exemplars was assessed by counting the number of expected pairings of stimuli found in the participant sorts.

While this was a reliable measure to identify similarity to an exemplar, it provided less useful in understanding the implications of the differences, that is, the unexpected pairings. It was observed that sorts of introductory students had many more unexpected pairings than did the sorts of the putative experts (Bissonnette, et al., 2017). This led Krieter et al. (2016) to evaluate the use of the edit distance metric to calculate a measure of proximal distance to the exemplar sorts for surface and deep factor categorizations. Using this measure they found that the proximal distance to an exemplar of deep factor categorization decreased as students attained progressive coursework milestones.

For the studies of computer science education, researchers designed the stimuli set to broadly investigate the conceptual representations about programming constructs across a large, diverse sample of introductory programming students (Sanders et al., 2005). Although this stimuli set was administered in two studies of different target populations [introductory (Sanders et al., 2005) and seniors (McCauley et al, 2005)] and the results analyzed in three additional papers (Deibel et al., 2005; Fossum & Haller, 2005; Murphy, et al., 2005) no references were found to preconceived notions of desired or expected categorizations. Indeed, in the originating study, the researchers concluded that given the analytical methods and tools available at that time, a rigorous categorization of the 1199 card sorts was cost and time prohibitive. In response to this restriction, the edit distance metric (Deibel et al., 2005) and the NMST measure of orthogonality (Fossum & Haller, 2005) were defined and evaluated.

Subsequent studies explored the potential of the new measures to facilitate categorization. By using cluster analysis to identifying structurally similar sorts, Deibel et al., 2005 were able to compare participant supplied category names and criterion

descriptions in subsets of the Sanders study dataset to identify several common categorization themes, such as *Easy or Hard to use*, *Concrete or Abstract*, and *Players, Formations, and Coaching*. McCauley et al. (2005) collected a smaller dataset of 291 card sorts from graduating seniors, and were able to conduct a rigorous content analysis to identify sixteen Content Analysis Groups, or CAGs. Eleven of these CAGs were believed to be organized around coherent themes. Using the NMST measure, the orthogonality of the sorts within each of these CAGs was calculated and confirmed that these eleven CAGs were composed of similarly structured sorts. These categorization themes included *Ease/Difficulty of use*, *Abstract/Concrete*, *Design/Software Engineering*, *Types of Programming Terms*, and *Entities and operations*. It is significant that these two studies using different datasets and different analytical measures and methods independently identified similar categorizations from the same stimuli set.

As described in Chapter 4, this current study identified cliques of categorizations within the sorts of participants considered most likely to have achieved the desired level of conceptual development. These cliques align well with the set of CAGs from McCauley et al. (2005). Structural exemplar sorts were then derived for several of these presumed deep factor categorizations. Measures of proximal distance to these exemplars were then calculated for participants' card sorts.

Results of the analysis for research question 3 replicated the finding from Kreiter et al. (2016) that the proximal distance to an exemplar of deep factor categorization decreased as students attained progressive coursework milestones. This finding suggests that proximal distances to exemplars of common categorizations for this stimuli set can

effectively differentiate conceptual development levels of students between, as well as within, cross-sections selected by achievement of coursework milestones.

Effects of programming experience and practice on proximity measures for differentiating conceptual development levels. As reported in Chapter 4, the current study analyzed variances in the edit distance between the participant card sorts to the exemplar sorts by categories of their programming experience (light, moderate, extensive). While statistically significant differences were found between participants reporting extensive experience versus light experience in the proximal distance of their card sorts to exemplars 21 and 185 (representing the *Types of Programming Terms* categorization), these differences trended in the direction opposite to that expected by the theoretical framework. The observed trend was that card sorts of participants with greater levels of experience were more distant from, rather than closer to, the exemplars. Due to this contrary finding, further analysis of the reported programming experiences of the population in this study was undertaken.

Since the prior analysis into research question 1 revealed an unequal distribution of experience levels between introductory, and non-introductory groups of participants with multiple card sorts ($n = 68$), a cross tabulation was prepared for categories of programming experience by categories of coursework achievement for all participants, including single card sorts ($n = 124$). The result is depicted in Figure 29. As noted earlier, the majority of the Introductory group reported extensive programming experiences, while the majority of the Completing group reported only light experience. These differences were then examined in a cross tabulation of programming experience by intended degree major as shown in Figure 30.

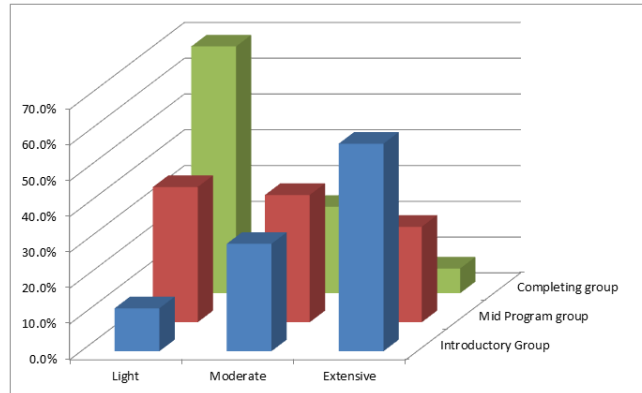


Figure 29. Percentages of programming experience categories per coursework achievement category (n = 124).

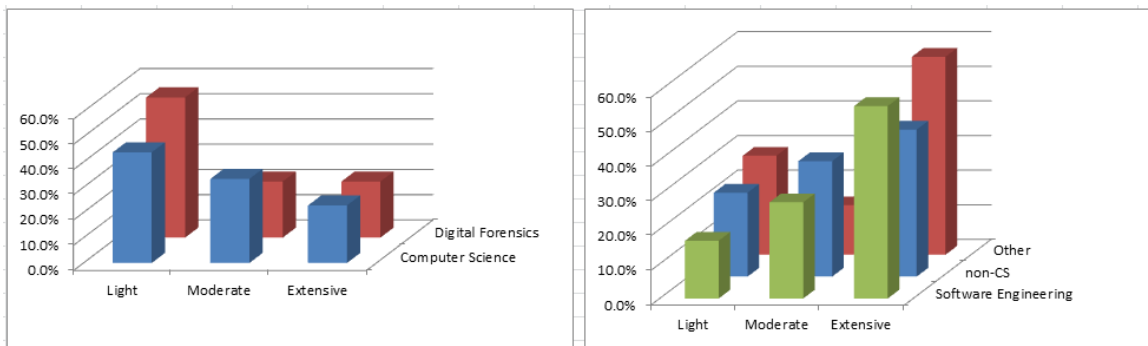


Figure 30. Percentage of experience level by intended majors.

The trend of decreasing percentages of programming experience seen in Figure 29 for the Completing group is found in the Figure 30 trends for participants majoring in digital forensics and computer science (left chart). However, the opposite trend, matching that of the Introductory group in Figure 29 is found in Figure 30 for software engineering technology and non-CS department majors (right chart). The non-CS department students are mostly Electronics and Computer Engineering Technology (ECET) majors who are not required to take as many hours of COSC courses as are CS department majors. This is reflected in the composition each category of coursework achievement by intended major as shown in Figure 31. While the majority of the Introductory group consists of the non-CS and software engineering technology majors, the percentage of students majoring in

computer science nearly doubles at mid-program, and is then predominant in the Completing group.

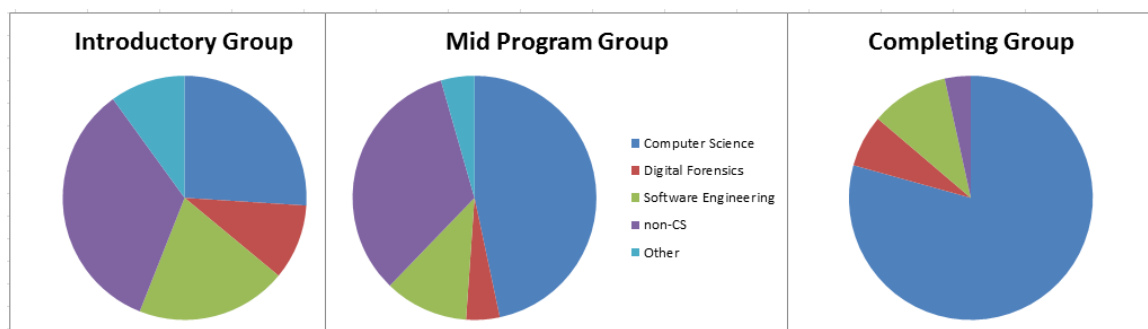


Figure 31. Percentage of intended majors for categories of coursework achievement.

One possible conclusion from these cross tabulations is that students drawn to degrees in software engineering technology and ECET are more prone to seek out the types of informal or job related programming experiences listed in the study questionnaire. Thus, a higher percentage of participants in the Introductory group report extensive programming experience. Due to the observed variations in composition of the cross-sections in terms of intended majors and experience levels, research question 4 was re-tested to determine if using only the group of introductory participants would alter the trend lines for the proximal distances to the exemplar sorts.

The trend lines from the original analysis using all participants are shown in Figure 32. Figure 33 shows the lines for the same exemplars for only the Introductory cross-section. In both figures the lines trend in the same direction. Therefore, it can be concluded that experience and practice with programming skills as captured by the study questionnaire does not support the expectation of the theoretical framework that experience fosters the desired conceptual growth.

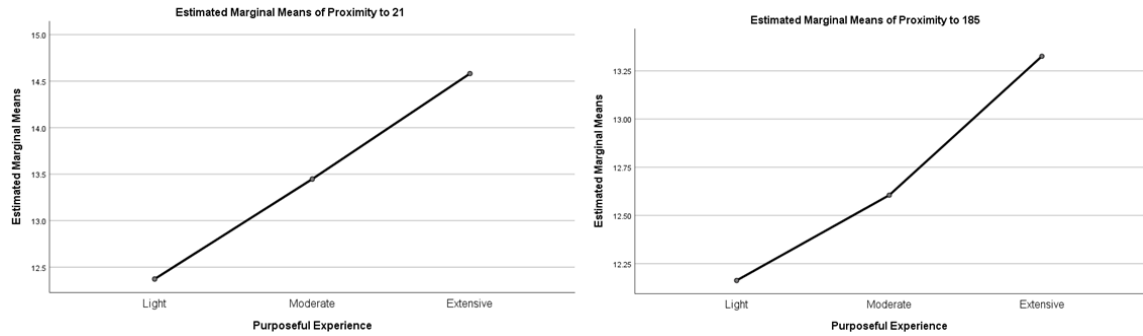


Figure 32. Proximal distances to Exemplars 21 and 185 (all participants).

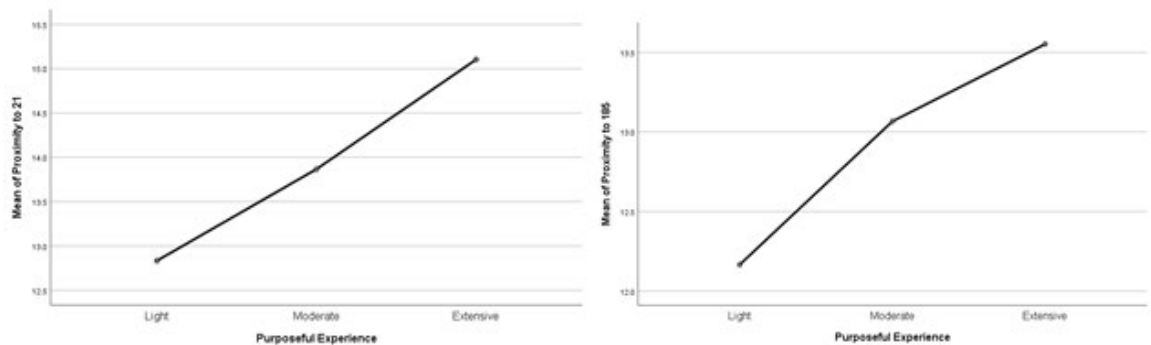


Figure 33. Proximal distances to Exemplars 21 and 185 (Introductory group).

This contrary finding may be explained by the nature of the exemplars used as the basis for the proximity measures. As noted in Chapter 4, both exemplar sorts 21 and 185 align with the *Types of Programming Terms* criterion as categorized by McCauley et al. (2005). Exemplar 21 in particular, shown below in Figure 34, reflects differentiation based upon computer science concepts that are only fully addressed in courses associated with later milestones of coursework achievement, such as Data Structures, Advanced Language Concepts, and Operating Systems. Therefore, programming experiences alone, as enumerated by the study questionnaire, appear to be insufficient for developing the conceptual representations expressed in Exemplar 21. For this deep-factor categorization criterion, the data supports a conclusion that formal instruction fosters the desired

conceptual development with greater efficacy than does additional programming practice and experience.

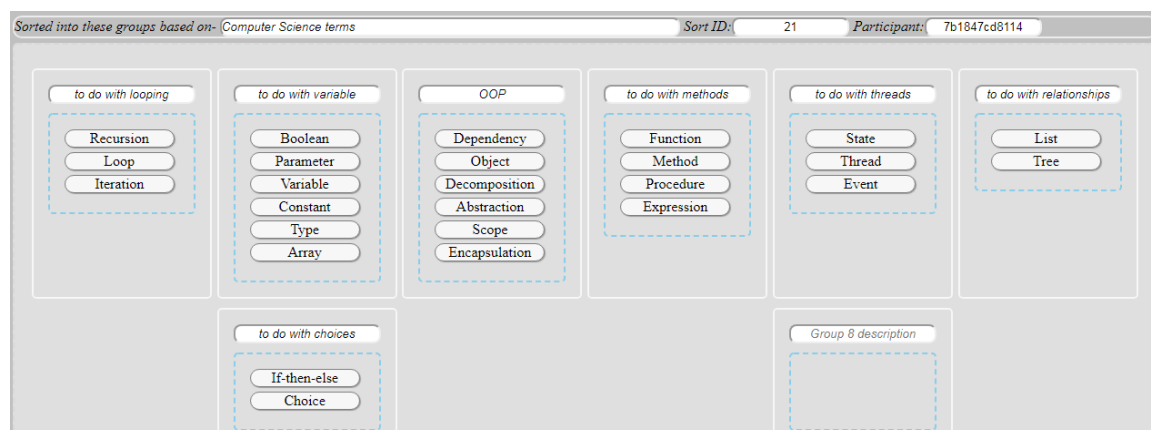


Figure 34. Exemplar sort 21 for the *Types of Programming Terms* categorization.

Conclusion

The results of this study successfully replicated findings of similar, prior studies of computer science students. For example, within the cross-section of completing students, the orthogonality of participant card sorts increased with each category of increase in academic performance. Also, collections of structurally similar card sorts were found to align with categorizations identified in earlier studies.

These findings provide the basis for responding to Denning’s call for the establishment of “guidelines for different skill levels of computational thinking” (2017, p. 36). Such a baseline of measures can be constructed from categorizations of elicited conceptual representations and associated exemplars. As shown in this and prior studies, students’ progress in development of computational thinking skills can be assessed, using their elicited representations, using multiple measures. The first two measures involve conceptual differentiation. Students grow in their ability to categorize in additional, but also more varied ways. Differentiated representations align with, or aggregate into, a

greater number of common categorizations. Therefore, differentiation might be measured as the number of common categorizations matched (McCauley et al., 2005). A second measure of differentiation is the orthogonality among the elicited card sorts (Fossum & Haller, 2005).

An additional measure of elicited conceptual development is the degree of similarity to an exemplar of an expected representation. The conceptual representations of novice learners often contain inaccuracies and simplifications which become better defined over time. The proximal edit distance metric from an exemplar quantifies how similar/dissimilar a student's representation is from the expected (Deibel et al., 2005). As students' conceptual representations progress, their card sorts become more similar to the desired exemplar. Therefore, establishing a baseline of measures for quantitative assessment of computational thinking skills should identify expected representations for a number of expected categorizations of card sorts. Eleven of the Content Analysis Groups identified by McCauley et al. (2005) appear to present reliable expectations for categorizations. Further research is needed to identify representative exemplars for each of these categorizations.

In this current study, measures of proximal distance to exemplars for one category were shown to be reliable differentiators of levels of conceptual development. However, conceptual development as elicited by these specific exemplars was found to be more influenced by instruction than by practice. This may be different for other categorizations of conceptual representation, such as Design versus Coding, where conceptual development may be more influenced by practice with the skill in accordance with the theoretical framework from Dreyfus and Dreyfus (1980). For example, results from

comparing the orthogonality measure of introductory to completing participants suggested that introductory students with greater levels of experience also had more differentiated card sorts. Therefore, follow-up studies should be conducted to identify exemplars of additional categorizations and to evaluate which of these benefit from increased levels of programming practice.

Exemplars should also be identified that are representative of students at earlier stages of their conceptual development. For example, card sorts for the dichotomous criterion *Items I know/Don't know* might be expected to change as a student progresses through the degree program. Since conceptual development often is not linear, exemplars at intermediate stages of development would provide a more accurate assessment than a single measure of distance from the desired end point.

Findings from this study further suggest that there should be expected differences in the target level of conceptual development for the various degree majors, such as ECET, and Software Engineering Technology, in addition to Computer Science. Therefore, exemplars should be identified that are representative of the desired level of conceptual development for that major.

Finally, this study used a repeated, open criterion card sorting activity with a stimuli set of 26 programming terms as the instrument for eliciting participants' organization of knowledge regarding computer programming concepts. A comparison with other studies' use of card sorting tasks suggests potential for improving aspects of the elicitation protocol for use with the baseline measures proposed above. An alternative protocol is the framed, as opposed to open, card sort in which the researcher prompts the participant to sort using a specific criterion (Rugg & McGeorge, 2005). However, framed

card sorts require the researcher to have identified criteria in advance. This was possible for the studies in the fields of biology and chemistry where students were given both open and framed activities. Card sorts obtained in both cases yielded similar results (Bissonnette, et al., 2017; Krieter et al., 2016). Given a set of desired categorizations for computational thinking skills as proposed above, framed card sorts could be used to elicit desired criteria from computer science students. This change would address an issue in this current study, where the use of the open card sort had an unexpected result of allowing too many instances of single card sorts. With simple modifications to the online tool for administration of the activity, it would be possible to prompt participants with a mix of open and framed sorting activities thereby eliminating single sort instances and ensuring adequate cases of diverse categorizations.

The objective of this study was to establish a baseline of computational thinking skill acquisition as an additional tool for evaluation of student progress toward the ABET student outcome 2: competency in the development of computer-based solutions to meet specified requirements (ABET, 2019). As compared to the common practice of qualitatively assessing programming assignments, the comparison of knowledge elicitation results against baseline indicators can yield objective, quantitative, computer-assessed measures of progress toward skill acquisition. Given an elicitation tool and a set of exemplar representations as proposed above, institutions could establish expected ranges of proximal distance measures to specific exemplars, selected according to particular categorizations, degree majors, and coursework milestones as evidence that students are meeting or exceeding the program learning objective for developing competency in the design and implementation of computer-based solutions.

Additionally, such measures would enable timely identification of those students not progressing as expected and would provide insights for the design of appropriate intervention measures. Such an outcome will be of benefit to the students and potentially to the prospective employers of the students as they may have greater assurance of students' competency as programmers and analytical problem solvers.

Beyond the specific focus on Computer Science Education, this study provides a research basis for transfer of new assessment approaches to other fields of education. One example is the Instructional Systems Design and Technology (ISDT) degree program.

Instructional Systems Design shares a development methodology with Software Development, i.e., computer programming. Goals are identified and agreed upon with various groups of stakeholders, and specified as measurable learning objectives to be attained. Teaching and Learning methods and materials are developed to achieve these learning objectives. Assessment methods and instruments are developed and utilized to evaluate attainment of the objectives.

The ISDT program at this university delves into the impact that information and communication technology has on this methodology for Instructional Systems Design. For Teaching and Learning, technology has introduced disruptive methods, media, and mediums. It is enabling shifts in paradigms between teachers and learners, and between instructional presentation and knowledge construction. Examples include blended or flipped classrooms and asynchronous communications.

For assessment, technology to date has mostly been used to offer enhancements to the test media and mediums. However, the methods of assessment have not yet been

disrupted by technology. Assessment tools, even when technology based, are still derived from the instruments of behaviorist educators.

Technology has begun to impact learning objectives through the reform initiatives for Computational Thinking and Coding for All. Furthermore, as students are gaining persistent, mobile access to commercial cognitive cloud services such as Siri, Alexa, and Cortana, there is a need to shift instructional objectives to the development of learners' ability to transfer knowledge, i.e., their cognitive skills, and away from requirements for them to retain and recall knowledge, i.e., cognitive concepts.

This study has demonstrated the feasibility of, and techniques for the quantitative and objective assessment of learners' development of high-order cognitive skills. These skills are required for transfers of knowledge which enable problem solving and solutions design. Assessment methods for these cognitive skills, such as those demonstrated in this study, will be necessary for educators and instructional designers to keep pace with the disruptions that technology is bringing to the Instructional Systems Design process.

REFERENCES

- ABET. (2019). *Criteria for accrediting computing programs, 2019-2020*. Retrieved from ABET: <https://www.abet.org/accreditation/accreditation-criteria/criteria-for-accrediting-computing-programs-2019-2020/>
- Adams, J., & Reed, D. A. (2015). Introducing young women to CS, and supporting advanced research environments. *Communications of the ACM*, 58(5), 10-11. doi:10.1145/2742480
- Ahadi, A., & Lister, R. (2013). Geek genes, prior knowledge, stumbling points and learning edge momentum: Parts of the one elephant? *Proceedings of the ninth annual international ACM conference on international computing education research* (pp. 123-128). ACM.
- Aho, A. V. (2012). Computation and computational thinking. *The Computer Journal*, 55(7), 832-835. doi:10.1093/comjnl/bxs074
- Ambrosio, A. P., Xavier, C., & Georges, F. (2014). Digital ink for cognitive assessment of computational thinking. *Frontiers in education conference (FIE), 2014 IEEE* (pp. 1-7). IEEE. doi:10.1109/FIE.2014.7044237
- Anderson, L. W., Krathwohl, D. R., Airasian, P. W., Cruikshank, K. A., Mayer, R. E., Pintrich, P. R., . . . Wittrock, M. C. (2001). *A taxonomy for learning, teaching and assessing: A revision of Bloom's Taxonomy of educational outcomes: Complete edition*. New York: Longman.
- Bell, R. (2011). Personal constructs. In L. Cohen, L. Manion, & K. Morrison (Eds.), *Research methods in education* (7th ed., pp. 496-509). London and New York: Routledge.

- Benner, P. (1982). From novice to expert. *The American Journal of Nursing*, 82(3), 402-407.
- Bergin, S., & Reilly, R. (2005). The influence of motivation and comfort-level on learning to program. In P. Romero, J. Good, A. Chaparro, & S. Bryant (Ed.), *17th Workshop of the Psychology of Programming Interest Group* (pp. 293-304). Sussex University. Retrieved from <http://eprints.teachingandlearning.ie/1770/2/Bergin%20and%20Reilly%202005.pdf>
- Bissonnette, S. A., Combs, E. D., Tanner, K. D., Nagami, P. H., Byers, V., Fernandez, J., . . . Smith, J. I. (2017). Using the Biology Card Sorting Task to measure changes in conceptual expertise during postscondary biology education. *CBE-Life Sciences Education*, 16(1), 1-15.
- Botha, F. (2016). Rethinking computational thinking. *Communications of the ACM*, 59(7), 8. doi:10.1145/2949401
- Brennan, K., & Resnick, M. (2012). New frameworks for studying and assessing the development of computational thinking. *Proceedings of the 2012 annual meeting of the American Educational Research Association* (pp. 1-25). Vancouver, Canada: AERA.
- Burleson, W. (2005). Developing creativity, motivation, and self-actualization with learning systems. *International Journal of Human-Computer Studies*, 63(4), 436-451. doi:10.1016/j.ijhcs.2005.04.007
- Campbell, P. J. (2016). Coding for all? *UMAP Journal*, 37(4), 333-337.

- Carraccio, C., Benson, B. J., Nixon, L. J., & Derstine, P. L. (2008). From the educational bench to the clinical bedside: Translating the Dreyfus developmental model to the learning of clinical skills. *Academic Medicine*, 83(8), 761-767.
doi:10.1097/ACM.0b013e31817eb632
- Casperson, M. E., & Kolling, M. (2009). STREAM: A first programming process. *ACM Transactions on Computing Education (TOCE)*, 9(1), 4.
doi:10.1145/1513593.1513597
- Casperson, M. E., Larsen, K. D., & Bennedsen, J. (2007). Mental models and programming aptitude. *ACM SIGCSE Bulletin*. 39, pp. 206-210. ACM.
- Chi, M. T., Feltovich, P. J., & Glaser, R. (1981). Categorization and representation of physics problems by experts and novices. *Cognitive Science*, 5(2), 121-152.
- Code.org. (2018). *About us*. Retrieved from Code.org: <https://code.org/about>
- Code.org. (2018a). *About us*. Retrieved from Code.org: <https://code.org/about>
- Code.org. (2018b). *Infographic source data*. Retrieved from Code.org: https://docs.google.com/document/d/1gySkItxiJn_vwb8HIIKNXqen184mRtzDX12cux0ZgZk/pub
- Code2College. (2017). *About*. Retrieved from code2college.org: <https://code2college.org/about>
- Cohen, L., Manion, L., & Morrison, K. (2011). Choosing a statistical test. In *Research methods in education*. London: Routledge.
- Cohen, L., Manion, L., & Morrison, K. (2011). *Surveys, longitudinal, cross-sectional and trend studies* (7th ed.). London: Routledge.

- Czerkawski, B., & Lyman, E. (2015). Exploring issues about computational thinking in higher education. *TechTrends: Linking Research & Practice to Improve Learning*, 59(2), 57-65. doi:10.1007/s11528-015-0840-3
- Dehnadi, S., & Bornat, R. (2006). The camel has two humps. *Middlesex University, UK*, 1-21. Retrieved from <http://eis.sla.mdx.ac.uk/research/PhDArea/saeed/paper1.pdf>
- Deibel, K., Anderson, R., & Anderson, R. (2005). Using edit distance to analyze card sorts. *Expert Systems*, 22(3), 129-138.
- Denning, P. (2017). Remaining trouble spots with computational thinking: Addressing unresolved questions concerning computational thinking. *Communications of the ACM*, 60(6), 33-39. doi:10.1145/2998438
- Dreyfus, H. L., Dreyfus, S. E., & Zadeh, L. A. (1987). Mind over machine: The power of human intuition and expertise in the era of the computer. *IEEE Expert*, 2(2), 110-111. doi:10.1109/MEX.1987.4307079
- Dreyfus, S. E., & Dreyfus, H. L. (1980). *A five-stage model of the mental activities involved in directed skill acquisition*. University of California, Berkley Operations Research Center.
- Fang, X. (2012). Application of the participatory method to the computer fundamentals course. In J. Luo (Ed.), *Affective Computing and Intelligent Interaction* (pp. 185-189). doi:10.1007/978-3-642-27866-2_23
- Fincher, S., & Tenenbergs, J. (2005). Making sense of card sorting data. *Expert Systems*, 22(3), 89-93. doi:10.1111/j.1468-0394.2005.00299.x
- Fossum, T., & Haller, S. (2005). Measuring card sort orthogonality. *Expert Systems*, 139-146.

- Garner, S. (2009). Learning to program from scratch. *2009 Ninth IEEE International Conference on Advanced Learning Technologies*, (pp. 451-452).
doi:10.1109/ICALT.2009.50
- Girls Who Code. (2018). *About us*. Retrieved from GirlsWhoCode:
<https://girlswhocode.com/about-us/>
- Gladwell, M. (2008). *Outliers: The story of success*. UK: Hachette.
- Google. (2018). *About*. Retrieved from Made w/ Code:
<https://www.madewithcode.com/about>
- Guzdial, M., & Morrison, B. (2016). Growing computer science education into STEM education discipline. *Communications of the ACM*, 59(11), 31-33.
doi:10.1145/3000612
- Hayes, J. R., & Simon, H. A. (1976). The understanding process: Problem isomorphs. *Cognitive Psychology*, 8, 165-190.
- Hewner, M. (2013). Undergraduate conceptions of the field of computer science. *Proceedings of the ninth annual international ACM conference on International computing education research* (pp. 107-114). ACM.
- IBM Corp. (2019, April). IBM® SPSS® Statistics 25 [Computer software]. Armonk, NY: IBM Corp. Retrieved from <https://www-01.ibm.com/support/docview.wss?uid=swg24043678>
- Irby, S. M., Phu, A. L., Borda, E. J., Haskell, T. R., Steed, N., & Meyer, Z. (2016). Use of a card sort task to assess students' ability to coordinate three levels of representation in chemistry. *Chemistry Education Research and Practice*, 17(2), 337-352.

- Johnston, H. C. (1976). Cliques of a graph- Variations on the Bron-Kerbosch algorithm. *International Journal of Computer & Information Sciences*, 5(3), 209-238.
- Kahneman, D., Slovic, P., & Tversky, A. (1982). *Judgement under uncertainty: Heuristics and Biases*. Cambridge University Press.
- Katsanos, C., Tselios, N., & Avouris, N. (2008). AutoCardSorter: Designing the information architecture of a web site using latent semantic analysis. *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (pp. 875-878). New York, NY: ACM.
- Kelly, G. (1955). *The psychology of personal constructs*. New York: Norton.
- Kirk, R. E. (1996). Practical significance: A concept whose time has come. *Educational and Psychological Measurement*, 56(5), 746-759.
- Koh, K. H., Basawapatna, A., Bennett, V., & Repenning, A. (2010). Towards the automatic recognition of computational thinking for adaptive visual language learning. *Visual Languages and Human-Centric Computing*, (pp. 59-66). doi:10.1109/VLHCC.2010.17
- Krieter, F. E., Julius, R. W., Bush, S. D., Scott, G. E., & Tanner, K. D. (2016). Thinking like a chemist: Development of a chemistry card-sorting task to probe conceptual expertise. *Journal of Chemical Education*, 93(5), 811-820.
- Lister, R. (2011). Ten years after the McCracken working group. *ACM Inroads*, 2(4), 18-19. doi:10.1145/2038876.2038882
- Lye, S. Y., & Koh, J. H. (2014). Review on teaching and learning of computational thinking through programming: What is next for K-12? *Computers in Human Behavior*, 41, 51-61. doi:10.1016/j.chb.2014.09.012

- Mason, A., & Singh, C. (2011). Assessing expertise in introductory physics using categorization task. *Physical Review Special Topics - Physics Education Research*, 7(2), 1-17. doi:10.1103/PhysRevSTPER.7.020110
- McCartney, R., Boustedt, J., Eckerdal, A., Sanders, K., & Zander, C. (2013). Can first-year students program yet?: A study revisited. *Proceedings of the Ninth Annual International ACM Conference on International Computing Education Research* (pp. 91-98). New York, NY: ACM. doi:10.1145/2493394.2493412
- McCartney, R., Boustedt, J., Eckerdal, A., Sanders, K., & Zander, C. (2017). Folk pedagogy and the Geek Gene: Geekiness quotient. *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education* (pp. 405-410). ACM.
- McCauley, R., Murphy, L., & Westbrook, S. (2005). What do successful computer science students know? An integrative analysis using card sort measures and content analysis to evaluate graduating students' knowledge of programming concepts. 22(3), 147-159. doi:10.1111/j.1468-0394.2005.00306.x
- McCracken, M., Almstrum, V., Diaz, D., Guzdial, M., Hagan, D., Kolikant, Y. B., . . . Wilusz, T. (2001). A multi-national, multi-institutional study of assessment of programming skills of first-year CS students. *Working Group Reports from ITiCSE on Innovation and Technology in Computer Science Education* (pp. 125-180). New York, NY: ACM. doi:10.1145/572133.572137
- Mertler, C. A., & Vannatta, R. A. (2013). *Advanced and multivariant statistical methods*. Glendale, CA: Pyrczak Publishing.

Murphy, L., McCauley, R., Westbrook, S., Fossum, T., Haller, S., Morrison, B. B., . . .

Anderson, R. E. (2005). A multi-institutional investigation of computer science seniors' knowledge of programming concepts. *Proceedings of the 36th SIGCSE technical symposium on computer science education* (pp. 510-514). New York, NY: ACM. doi:10.1145/1047344.1047505

Newell, A., & Simon, H. A. (1972). *Human problem solving*. Englewood Cliffs, NJ: Prentice-Hall.

Pallant, J. (2001). *SPSS Survival Manual*. Maidenhead: Open University Press and McGraw-Hill Education.

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

Patitsas, E., Berlin, J., Craig, M., & Easterbrook, S. (2016). Evidence that computer science grades are not bimodal. *Proceedings of the 2016 ACM conference on international computing education research* (pp. 113-121). ACM.

Paul, A. M. (2016). The coding revolution. *Scientific American*, 315(2), 42-49. doi:10.1038/scientificamerican0816-42

Pena, A. (2010). The Dreyfus model of clinical problem-solving skills acquisition: A critical perspective. *Medical Education Online*, 15(1). doi:10.3402/meo.v15i0.4846

Petrie, H., Power, C., Cairns, P., & Seneler, C. (2011). Using card sorts for understanding website information architectures: Technological, methodological and cultural issues. *Human-Computer Interaction - INTERACT 2011* (pp. 309-322). Berlin, Heidelberg: Springer. doi:10.1007/978-3-642-23768-3_26

- Piaget, J. (1969). *The psychology of the child*. New York: Basic Books.
- Polanyi, M. (1966). *The Tacit Dimension*. Chicago: The University of Chicago Press.
- Precht, B. S., Szwillus, G., & Domik, G. (2014). *Edit distance analysis of card sorting experiments*. (Master's Thesis). University of Paderborn.
- Robins, A. (2010). Learning edge momentum: A new account of outcomes in CS1. *Computer Science Education*, 20(1), 37-71.
- Robins, A., Rountree, J., & Rountree, N. (2003). Learning and teaching programming: A review and discussion. *Computer Science Education*, 13(2), 137-172.
doi:10.1076/csed.13.2.137.14200
- Rugg, G., & McGeorge, P. (2005). The sorting techniques: A tutorial paper on card sorts, picture sorts and item sorts. *Expert Systems*, 22(3), 94-97. doi:10.1111/j.1468-0394.2005.00300.x
- Rugg, G., Corbridge, C., Major, N. P., Burton, A. M., & Shadbolt, N. R. (1992). A comparison of sorting techniques in knowledge acquisition. *Knowledge Acquisition*, 4(3), 279-291.
- Sam Houston State University. (2017). *Department of Computer Science*. Retrieved from Sam Houston State University:
<http://catalog.shsu.edu/undergraduate/colleges-academic-departments/science-and-engineering-technology/computer-science/>
- Sanders, K., Fincher, S., Bouvier, D., Lewandowski, G., Morrison, B., Murphy, L., . . . Zander, C. (2005). A multi-institutional, multinational study of programming concepts using card sort data. *Expert Systems*, 22(3), 121-128.
doi:10.1111/j.1468-0394.2005.00303.x

- Scott, J., & Bundy, A. (2015). Creating a new generation of computational thinkers. *Communications of the ACM*, 58(12), 37-40. doi:10.1145/2791290
- Selby, C. (2015). Relationships: Computational thinking, pedagogy of programming, and Bloom's taxonomy. *The 10th Workshop in Primary and Secondary Computing Education (WipSCE)* (pp. 80-87). London: ACM. doi:10.1145/2818314.2818315
- Shingala, R. E., & Rajyaguru, D. A. (2015). Comparison of post hoc tests for unequal variance. *International Journal of New Technologies in Science and Engineering*, 2(5), 22-33.
- Simon, D. P., & Simon, H. A. (1978). Individual differences in solving physics problems. In R. S. Siegler (Ed.), *Children's thinking: What develops?* Hillsdale, NJ: Erlbaum.
- Smith, M. U. (1990). Knowledge structures and the nature of expertise in classical genetics. *Cognition and Instruction*, 7(4), 287-302.
doi:10.1207/s1532690xci0704_1
- Upchurch, L., Rugg, G., & Kitchenham, B. (2001). Using card sorts to elicit web page quality attributes. *IEEE Software*, 18(4), 84-89. doi:10.1109/MS.2001.936222
- Voogt, J., Fisser, P., Good, J., Mishra, P., & Yadav, A. (2015). Computational thinking in compulsory education: Towards an agenda for research and practice. *Education and Information Technologies*, 20(4), 715-728. doi:10.1007/s10639-015-9412-6
- Wing, J. M. (2006). Computational thinking. *Communications of the ACM*, 49(3), 33-35. doi:10.1145/1118178.1118215
- Wing, J. M. (2008). Computational thinking and thinking about computing. *Philosophical Transactions of the Royal Society of London A: Mathematical,*

Physical and Engineering Sciences, 366(1881), 3717-3725.

doi:10.1098/rsta.2008.0118

Zhong, B., Wang, Q., Chen, J., & Li, Y. (2016). An exploration of three-dimensional integrated assessment for computational thinking. *Journal of Educational Computing Research*, 53(4), 562-590. doi:10.1177/0735633115608444

APPENDIX A

Card Sort Online Instrument

The Card Sort online instrument will appear similar to the following example created with the Proven by Client product.

The participant will first receive instructions on how to perform the sort as shown in Figure 35.

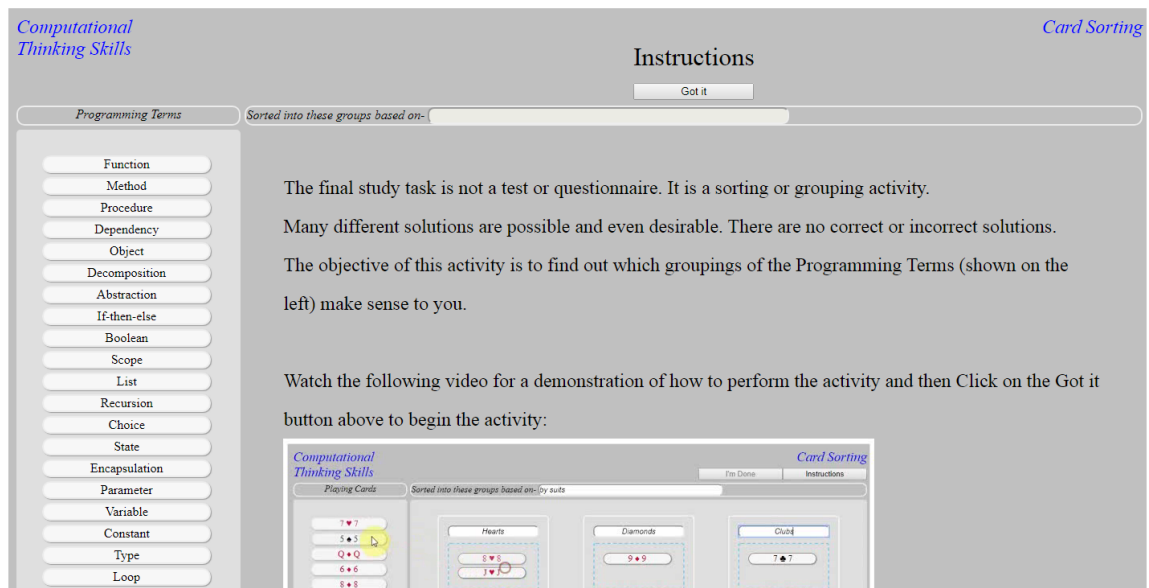


Figure 35. Card sort instrument screen 1. The initial screen of the online card sort instrument displays the instructions for completing the task to the participant.

Then the set of stimuli, that is, the cards, are presented. With this tool, they are presented in a vertical list down the left hand side of the screen as shown in Figure 36.

The screenshot shows a web-based card sorting instrument. On the left, a vertical list of programming terms is provided as stimuli: Boolean, Loop, Method, Tree, Variable, Recursion, Event, Decomposition, Procedure, Array, Type, Parameter, Function, List, Object, Scope, Encapsulation, State, If-then-else, Iteration, Dependency, Constant, Expression, Choice, Thread, and Abstraction. At the top right, there are buttons for 'Move Remainder' and 'Submit'. Below the title 'Card Sorting', there is a section for sorting criteria: 'Sorted into these groups based on-' followed by an input field for 'Optional description of sorting criterion'. On the right side, there is a box for 'Group 1 description' with a dashed rectangular area below it for placing cards. The bottom right corner of the interface is credited to 'M. Earnest Research'.

Figure 36. Card sort instrument screen 2. The starting point for each sort displays the stimuli set down the left-hand side of the screen with the remainder of the screen being blank.

The participant begins creating categories and groups the cards into the categories by dragging cards from the list to the right into new, or existing group boxes as shown in Figure 37. As cards are dropped into new categories, the participant is able to provide a label for the category.

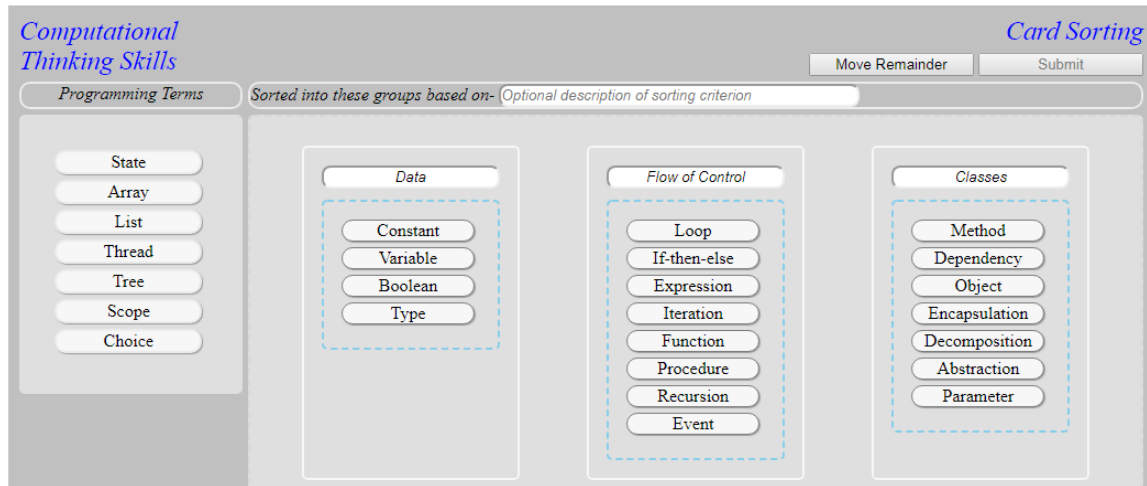


Figure 37. Card sort instrument screen 3. Three categories: data, flow of control, and classes have been created with items from the stimuli-set dropped into them..

As cards are categorized, they are removed from the vertical column on the left hand side as shown in comparing Figure 38 to Figure 39.

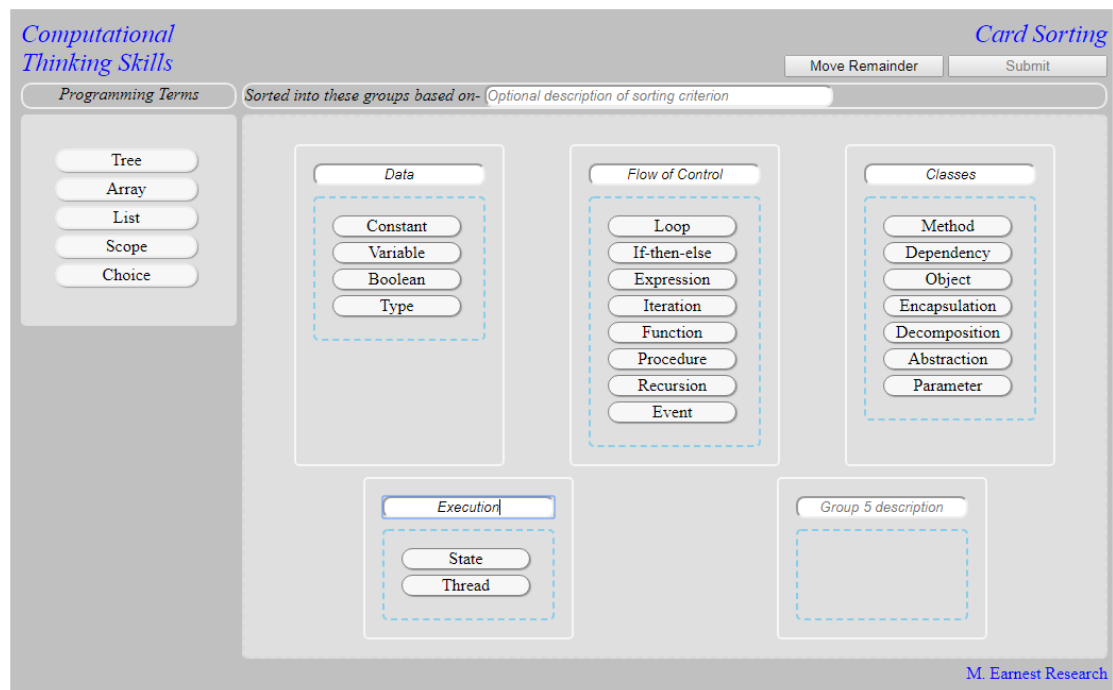


Figure 38. Card sort instrument screen 4. Twenty-one cards have been sorted into the four categories of data, flow of control, classes, and execution.

Computational Thinking Skills

Card Sorting

Move Remainder Submit

Sorted into these groups based on- Optional description of sorting criterion

Choice

Data

- Constant
- Variable
- Boolean
- Type
- Scope

Flow of Control

- Loop
- If-then-else
- Expression
- Iteration
- Function
- Procedure
- Recursion
- Event

Classes

- Method
- Dependency
- Object
- Encapsulation
- Decomposition
- Abstraction
- Parameter

Execution

- State
- Thread

Data Structures

- Array
- List
- Tree

Group 6 description

M. Earnest Research

Figure 39. Card sort instrument screen 5. Twenty-five cards have been sorted into five categories of data, flow of control, classes, data structures, and execution.

As shown in Figure 39, the participant has been able to categorize all but one of the cards. However, the instructions require the participant to place all cards in a group, that is, leave no cards in the Card List.

The user is prompted to create groups called “Don’t know”, or “Not applicable” to capture the remaining cards as shown in Figure 40.

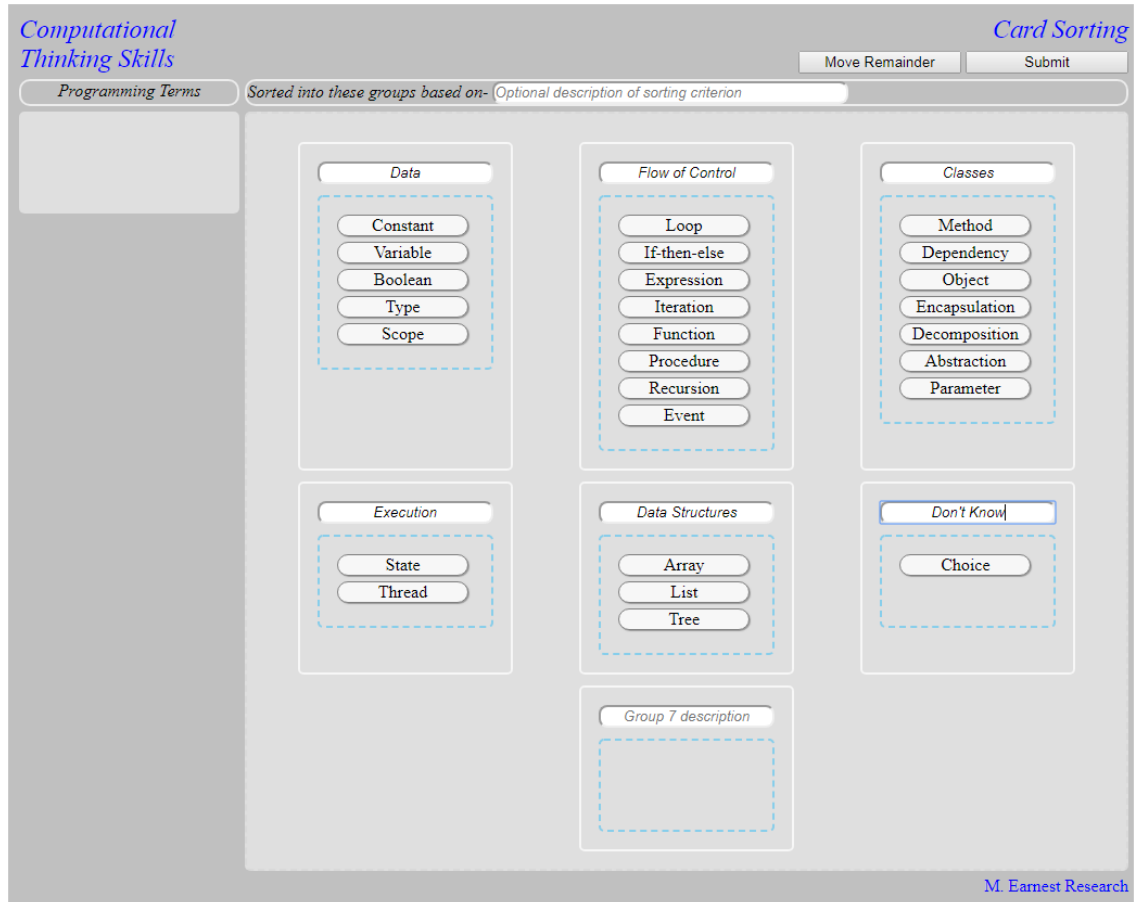


Figure 40. Card sort instrument screen 6. All cards sorted into the five user-created categories and an additional category for “Don’t Know”.

This particular sort is now completed. Next, the tool will collect and save the categories and their labels along with the cards in each group.

The stimulus set is randomly re-ordered, and the participant is asked to sort the cards again based on a new criterion.

APPENDIX B

Data Collection Questionnaire

Research Questionnaire

Thank you for participating in this research activity. This questionnaire is the first step in the activity and is designed to collect information that will be used to help analyze the data gathered by the following step. Please answer each question as accurately as possible and to the best of your knowledge. All information you enter will be treated as confidential, but you may choose to not respond to any question.

1. What is your current university classification level?

Mark only one oval.

- ☐ Freshman
- ☐ Sophomore
- ☐ Junior
- ☐ Senior
- ☐ Post Graduate

2. What is your intended major/program?

Mark only one oval.

- ☐ CS
- ☐ Software Engineering
- ☐ Digital and Cyber Forensic Engineering
- ☐ CS Minor
- ☐ Masters
- ☐ Other: _____

3. When you are given a programming assignment for a current or recent CS course here at SHSU, what is your usual experience in completing the assignment?

Mark only one oval.

- ☐ Not a problem. I can complete on my own in less than 2 hours.
- ☐ Minor problems with syntax or logic bugs. I can complete on my own in less than 3 hours.
- ☐ I think I get it, but I struggle to get the code to work. It takes me more than 3 hours to complete on my own.
- ☐ I often need to get help to write the code and get it to work.
- ☐ Some get it, and some don't. I don't get it. I hate coding assignments.

4. How would you rate your competence as a programmer?

Mark only one oval.

- ☐ I just don't get it. Computer science is OK, but I can't code.
- ☐ I am just beginning to learn
- ☐ I am confident that I can work through the usual challenges to complete my programs
- ☐ I enjoy programming and seem to have fewer problems than most other students
- ☐ I am really good and other students come to me for help

Prior learning in programming / coding

The questions in this section ask you to recall if you have had any formal or informal training in programming or coding PRIOR to beginning your CS coursework here at SHSU.

5. **Have you ever programmed micro-devices like a Raspberry Pi or Arduino (perhaps as part of a robotics kit)?**

Mark only one oval.

- ☐ Yes
☐ No

6. **Did you have any formal school training in programming or coding in the following grades? (check any/all that apply; leave blank if none apply)**

Check all that apply.

- ☐ Elementary (Grades K - 5)
☐ Middle (Grades 6 - 8)
☐ Grade 9
☐ Grade 10
☐ Grade 11
☐ Grade 12
☐ Community college
☐ Other college

7. **Have you received any formal training in programming or coding as part of a job? (Formal as in you attended a class, or were required to study and pass a test)**

Mark only one oval.

- ☐ Yes
☐ No

8. **Have you taken any informal training in programming or coding? (Informal as in self-study / self-improvement)**

Mark only one oval.

- ☐ Yes, for a job
☐ Yes, for self-improvement or personal interest
☐ No

Programming / coding experiences

This section asks about your experiences with programming or coding projects. In answering, please consider work you have done as part of formal and informal learning both before beginning your CS course of study at SHSU AND as part of your SHSU CS coursework. For these questions the responses range from examples requiring the least experience to those requiring the most experience. Please select the one response in each question whose examples best represent your level of experience.

9. How much experience do you have programming micro devices like a Raspberry Pi or Arduino?

Mark only one oval.

- ☐ None
- ☐ Had some exposure as part of a team
- ☐ Did it once, a long time ago
- ☐ Have done it, could do it again
- ☐ Multiple experiences, very comfortable with it

10. What abilities do you have to create web pages, and work with HTML/CSS/Javascript?

Mark only one oval.

- ☐ Have never done it
- ☐ Can create static web pages
- ☐ Can read and modify HTML
- ☐ Can create and modify CSS
- ☐ Can write javascript for use with HTML

11. What abilities do you have to create web sites?

Mark only one oval.

- ☐ Have never done it
- ☐ Can create a multi-page static site
- ☐ Can create sites with tools like Google Sites, Joomla, Drupal, etc.
- ☐ Can use php, ruby, or node.js on the backend to respond to web forms
- ☐ Can create dynamic pages that access cloud service APIs from Google, AWS, Microsoft, etc.

12. What experience do you have with github or similar code project services?

Mark only one oval.

- ☐ None
- ☐ Have used it to retrieve an application or library
- ☐ Have used it to search for and retrieve sample code
- ☐ Have established my own private code library
- ☐ Have used it for team code development and maintenance

Demographics

This section asks for personal information for the purpose of establishing demographic classifications. Answers you provide to these questions will only be reported in the aggregate. That is, your individual answers will be kept confidential and will NOT be separately identified, listed, or reported. You may elect not to answer these questions, however, providing answers will benefit the overall analysis of the data.

13. Year you were born (provide a number between 1940 and 2003):

14. The gender you identify with:

Mark only one oval.

☐ Female

☐ Male

☐ Prefer not to say

☐ Other:

15. The Ethnicity you identify with (such as, Asian, Black, Hispanic, White, etc.):

APPENDIX C

Statistical Analysis for Question 1

Research Question 1:

Is there a relationship between categories of coursework achievement (Introductory, Mid-Program, Completing) and categories (High, Low, or Zero) of card sort orthogonality?

Null and Alternative Hypotheses

H₀: There is no statistically significant relationship between categories of coursework achievement and categories of card sort orthogonality.

H_a: There is a statistically significant relationship between categories of coursework achievement and categories of card sort orthogonality.

Assumption Testing

All assumptions were met. Expected frequency counts for each category are greater than 5 as shown in the cross-tabulation table (Figure 41).

Results of Relationship Testing

A chi-square was calculated to determine the relationship between categories of coursework achievement and categories of card sort orthogonality as shown in Figure 42. No significant relationship was found ($\chi^2(4) = 8.65, p > .05$). There was no statistically significant relationship between categories of coursework achievement and categories of card sort orthogonality.

Coursework Achievement Category * Orthogonality Crosstabulation

			Orthogonality			
			High	Low	Zero	Total
Coursework	Introductory	Count	12	8	30	50
Achievement Category		Expected Count	14.1	13.3	22.6	50.0
		% within Coursework Achievement Category	24.0%	16.0%	60.0%	100.0%
		% within Orthogonality	34.3%	24.2%	53.6%	40.3%
		% of Total	9.7%	6.5%	24.2%	40.3%
	Mid	Count	14	14	17	45
Program		Expected Count	12.7	12.0	20.3	45.0
		% within Coursework Achievement Category	31.1%	31.1%	37.8%	100.0%
		% within Orthogonality	40.0%	42.4%	30.4%	36.3%
		% of Total	11.3%	11.3%	13.7%	36.3%
	Completing	Count	9	11	9	29
		Expected Count	8.2	7.7	13.1	29.0
		% within Coursework Achievement Category	31.0%	37.9%	31.0%	100.0%
		% within Orthogonality	25.7%	33.3%	16.1%	23.4%
		% of Total	7.3%	8.9%	7.3%	23.4%
Total		Count	35	33	56	124
		Expected Count	35.0	33.0	56.0	124.0
		% within Coursework Achievement Category	28.2%	26.6%	45.2%	100.0%
		% within Orthogonality	100.0%	100.0%	100.0%	100.0%
		% of Total	28.2%	26.6%	45.2%	100.0%

Figure 41. Cross-tabulation of categories of Coursework Achievement (Introductory, Mid-program, Completing) by categories of Orthogonality (High, Low, Zero).

Chi-Square Tests			
	Value	df	Asymptotic Significance (2-sided)
Pearson Chi-Square	8.647 ^a	4	.071
Likelihood Ratio	8.792	4	.067
Linear-by-Linear Association	3.890	1	.049
N of Valid Cases	124		

a. 0 cells (0.0%) have expected count less than 5. The minimum expected count is 7.72.

Figure 42. Chi-square testing the relationship of categories of Coursework Achievement (Introductory, Mid-program, Completing) with categories of Orthogonality (High, Low, Zero).

Examination of the counts and percentages indicated that the frequency of zero NMST values, representing only single sort submission, decreased as students progressed in their coursework, with 60% of Introductory students submitting single sorts, while 38% of Mid-program students, and 31% of Completing students submitted single sorts.

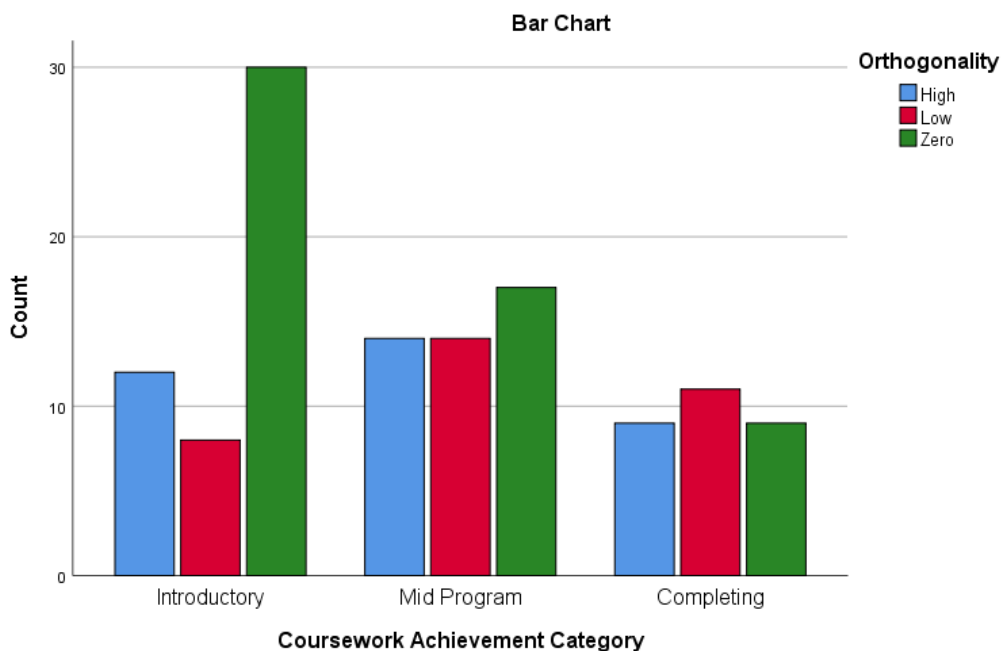


Figure 43. Frequency counts of categories of orthogonality (High, Low, Zero) by categories of coursework achievement (Introductory, Mid-program, Completing).

Comparable numbers of participants submitted multiple card sorts among the three categories of students: 20 each for Introductory and Completing students and 28 for Mid-program students. NMST values were calculated for participants and categorized as either High or Low. High NMST values, denoting greater differentiation among submitted card sorts, outnumbered low NMST values among the Introductory students (24% vs 16%), while this relationship was reversed among the Completing students (31% high vs 38% low). Mid program students had an equal percentage of high and low NMST values (31%).

The effect size determined by Cramer's V was .19 (Figure 44). This indicates that a small association was found between whether a student submitted card sorts with High, Low, or Zero NMST values and whether the student was at the Introductory, Mid-program, or Completing milestone in their computer science coursework.

Symmetric Measures			
		Value	Approximate Significance
Nominal by Nominal	Phi	.264	.071
	Cramer's V	.187	.071
N of Valid Cases		124	

Figure 44. Cramer's V determination of effect size for Chi-square.

APPENDIX D

Statistical Analysis for Question 2

Research Question 2

- A. Is there a difference in participant sort proximities to the probe sorts (6, 15, 21, 84, 85, 185, 193, 201) between members of collection 3 (membership = 0) and collection 13 (membership = 1)?
- B. How many reliable and interpretable structural exemplar sorts are derivable from the collections of sorts produced by those participants most likely to have attained the desired level of programing skill?

Null and Alternative Hypotheses

H_{A0} = There is no difference in participant sort proximities to the probe sorts (6, 15, 21, 84, 85, 185, 193, 201) between members of collection 3 and collection 13.

H_{Aa} = There is a difference in participant sort proximities to the probe sorts (6, 15, 21, 84, 85, 185, 193, 201) between members of collection 3 and collection 13.

H_{B0} = No reliable and interpretable structural exemplar sorts are derivable from the collections of sorts produced by those participants most likely to have attained the desired level of programing skill.

H_{Ba} = Reliable and interpretable structural exemplar sorts are derivable from the collections of sorts produced by those participants most likely to have attained the desired level of programing skill.

Assumption Testing for Independent Samples *t*-tests

There are no missing data values among membership in Collection 13 groups (Yes, No) for each of the sets of proximity values to the exemplar sorts categorized by (Figure 45). The groups are independent as the membership in collections 3 (membership = 0) and 13 (membership = 1) is mutually exclusive. The dependent variables are the scalar edit distance between participant sorts and each exemplar sort.

Case Processing Summary

		Valid		Cases Missing		Total	
	membership	N	Percent	N	Percent	N	Percent
Exemplar 6	0	32	100.0%	0	0.0%	32	100.0%
	1	23	100.0%	0	0.0%	23	100.0%
Exemplar 15	0	32	100.0%	0	0.0%	32	100.0%
	1	23	100.0%	0	0.0%	23	100.0%
Exemplar 21	0	32	100.0%	0	0.0%	32	100.0%
	1	23	100.0%	0	0.0%	23	100.0%
Exemplar 185	0	32	100.0%	0	0.0%	32	100.0%
	1	23	100.0%	0	0.0%	23	100.0%
Exemplar 193	0	32	100.0%	0	0.0%	32	100.0%
	1	23	100.0%	0	0.0%	23	100.0%
Exemplar 201	0	32	100.0%	0	0.0%	32	100.0%
	1	23	100.0%	0	0.0%	23	100.0%
Sort 84	0	32	100.0%	0	0.0%	32	100.0%
	1	23	100.0%	0	0.0%	23	100.0%
Sort 85	0	32	100.0%	0	0.0%	32	100.0%
	1	23	100.0%	0	0.0%	23	100.0%

Figure 45. Case Process Summary for the membership groups for the set of proximity values to the exemplar sorts.

Three cases of outliers were found among participant sort proximities to exemplar 6 in collection 3 (membership = 0). No other significant outliers exist (Figure 46).

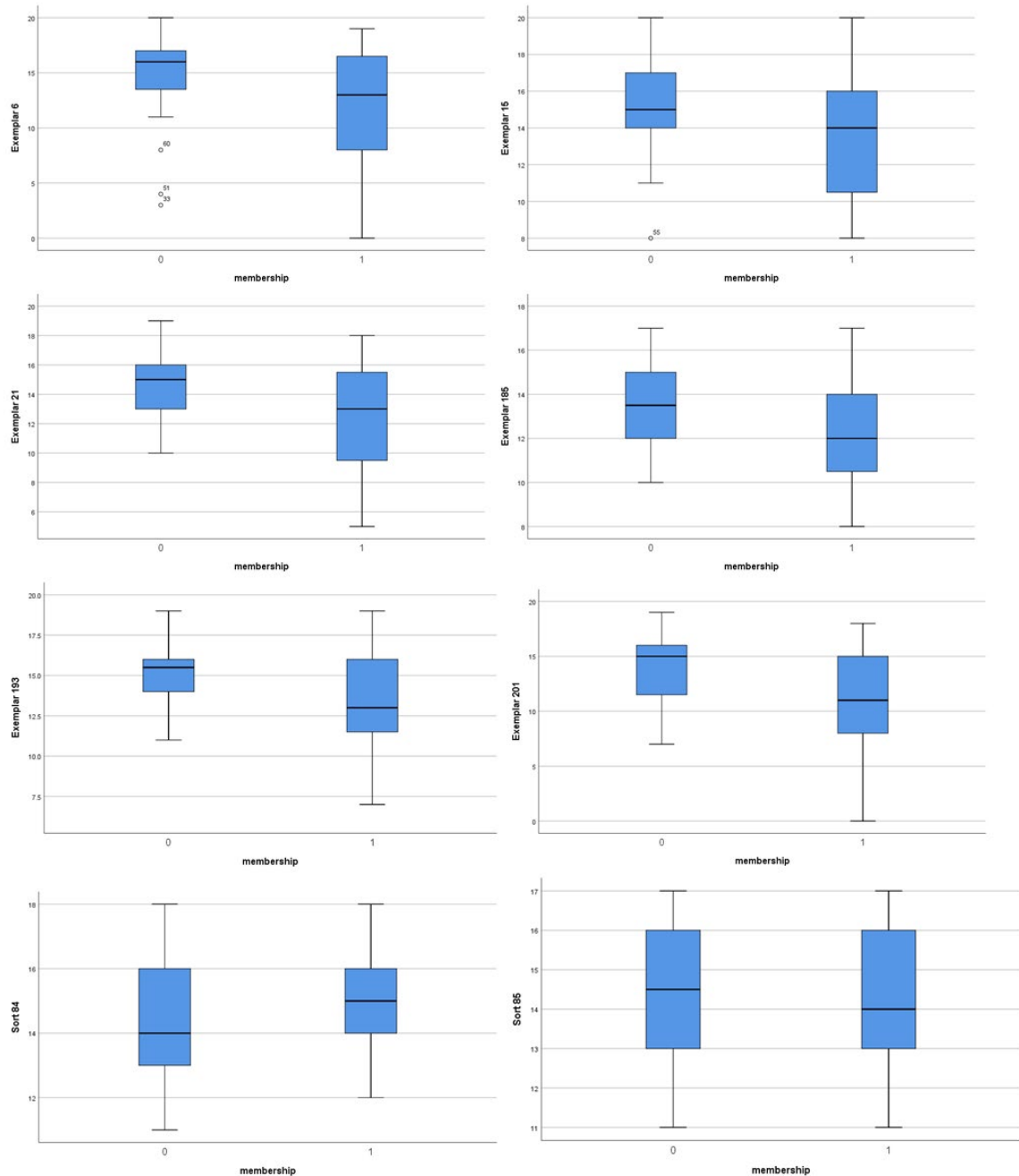


Figure 46. Box and whiskers plots for memberships groups for proximity to the exemplar sorts.

Inspection of Kolmogorov-Smirnov statistics (Figure 47) indicated that proximities to each of the eight probe sorts for members of collection 13 (membership = 1) and proximities to sorts 85, 185, and 201 for members of collection 3 (membership =

0) had significance $> .05$ indicating normal frequency distributions. Kolmogorov-Smirnov statistics for participant sort proximities to sorts 6, 15, 21, 84, and 193 for members of collection 3 were $< .05$. However, calculated z-scores (Figure 48) for all distributions except for proximities to sort 6 fell within ± 3 . Based on the Kolmogorov-Smirnov and z-scores, distributions for proximity to all sorts except sort 6 were assumed to be normal. Statistical analysis was conducted for the remaining sorts.

Levene's test for homogeneity (Figure 50) was found to be significant, indicating homogeneity of variance, when comparing groups for proximity to sorts 84, 85, and 185, but not significant when comparing groups for proximity to sorts 6, 15, 21, 193, or 201. Results of t-tests were interpreted according.

Tests of Normality

		Kolmogorov-Smirnov ^a			Shapiro-Wilk		
	membership	Statistic	df	Sig.	Statistic	df	Sig.
Exemplar 6	0	.185	32	.007	.846	32	.000
	1	.150	23	.194	.925	23	.084
Exemplar 15	0	.186	32	.006	.957	32	.234
	1	.112	23	.200*	.947	23	.259
Exemplar 21	0	.171	32	.018	.965	32	.382
	1	.121	23	.200*	.946	23	.243
Exemplar 185	0	.131	32	.172	.954	32	.187
	1	.115	23	.200*	.972	23	.743
Exemplar 193	0	.174	32	.015	.950	32	.147
	1	.116	23	.200*	.961	23	.480
Exemplar 201	0	.144	32	.089	.935	32	.053
	1	.137	23	.200*	.943	23	.209
Sort 84	0	.182	32	.009	.947	32	.115
	1	.163	23	.117	.958	23	.418
Sort 85	0	.132	32	.166	.948	32	.124
	1	.150	23	.196	.935	23	.141

*. This is a lower bound of the true significance.

a. Lilliefors Significance Correction

Figure 47. Test of Normality for the membership groups for the set of proximity values to the exemplar sorts.

Z-scores:

	Membership	Statistic	Std Err	Z score
0	S 6	-1.607	0.414	-3.882
	K 6	3.109	0.809	3.843
1	S 6	-0.668	0.481	-1.389
	K 6	-0.457	0.935	-0.489
0	S 15	-0.475	0.414	-1.147
	K 15	0.756	0.809	0.934
1	S 15	0.174	0.481	0.362
	K 15	-0.955	0.935	-1.021
0	S 21	0.057	0.414	0.138
	K 21	-0.336	0.809	-0.415
1	S 21	-0.304	0.481	-0.632
	K 21	-0.933	0.935	-0.998
0	S 185	-0.198	0.414	-0.478
	K 185	-0.749	0.809	-0.926
1	S 185	0.192	0.481	0.399
	K 185	-0.682	0.935	-0.729
0	S 193	-0.265	0.414	-0.640
	K 193	-0.097	0.809	-0.120
1	S 193	-0.12	0.481	-0.249
	K 193	-0.346	0.935	-0.370
0	S 201	-0.341	0.414	-0.824
	K 201	-0.81	0.809	-1.001
1	S 201	-0.394	0.481	-0.819
	K 201	-0.349	0.935	-0.373
0	S 84	0.03	0.414	0.072
	K 84	-0.708	0.809	-0.875
1	S 84	0.073	0.481	0.152
	K 84	-0.505	0.935	-0.540
0	S 85	-0.16	0.414	-0.386
	K 85	-0.678	0.809	-0.838
1	S 85	-0.12	0.481	-0.249
	K 85	-0.821	0.935	-0.878

Figure 48. Z-Scores for membership groups for proximity to each of the exemplar sorts.

Test of Homogeneity of Variance

		Levene Statistic	df1	df2	Sig.
Exemplar 6	Based on Mean	4.663	1	53	.035
	Based on Median	3.547	1	53	.065
	Based on Median and with adjusted df	3.547	1	52.996	.065
	Based on trimmed mean	4.573	1	53	.037
Exemplar 15	Based on Mean	4.662	1	53	.035
	Based on Median	4.354	1	53	.042
	Based on Median and with adjusted df	4.354	1	50.334	.042
	Based on trimmed mean	4.611	1	53	.036
Exemplar 21	Based on Mean	10.490	1	53	.002
	Based on Median	8.817	1	53	.004
	Based on Median and with adjusted df	8.817	1	39.983	.005
	Based on trimmed mean	10.379	1	53	.002
Exemplar 185	Based on Mean	1.426	1	53	.238
	Based on Median	1.267	1	53	.265
	Based on Median and with adjusted df	1.267	1	48.998	.266
	Based on trimmed mean	1.376	1	53	.246
Exemplar 193	Based on Mean	6.574	1	53	.013
	Based on Median	5.138	1	53	.028
	Based on Median and with adjusted df	5.138	1	42.680	.029
	Based on trimmed mean	6.632	1	53	.013
Exemplar 201	Based on Mean	3.987	1	53	.051
	Based on Median	3.558	1	53	.065
	Based on Median and with adjusted df	3.558	1	52.964	.065
	Based on trimmed mean	4.097	1	53	.048
Sort 84	Based on Mean	.908	1	53	.345
	Based on Median	.455	1	53	.503
	Based on Median and with adjusted df	.455	1	49.598	.503
	Based on trimmed mean	.911	1	53	.344
Sort 85	Based on Mean	.244	1	53	.623

Figure 49. Test of Homogeneity of Variance for the membership groups for the set of proximity values to the exemplar sorts.

Summary of Assumption Testing for independent samples t -tests: The assumptions of dependent Interval data and Independence for the membership in collections 3 (membership = 0) and 13 (membership = 1) are met for proximity to the sorts 6, 15, 21, 84, 85, 185, 193, and 201. Three outlying cases found in the proximities to sort 6 for members of collection 3 (membership = 0) were removed from the dataset and the distributions re-examined (Figure 50) and found to be normal, but with unequal variances.

Based upon the Kolmogorov-Smirnov statistics and z-scores, all distributions were assumed to be normal. Variances between groups were found to be equal for proximities to sorts 84, 85, and 185, but unequal for proximity to sorts 6, 15, 21, 193, and 201.

Case Processing Summary

		Valid		Cases Missing		Total	
	membership	N	Percent	N	Percent	N	Percent
Exemplar 6	0	29	100.0%	0	0.0%	29	100.0%
	1	23	100.0%	0	0.0%	23	100.0%

Tests of Normality

		Kolmogorov-Smirnov ^a			Shapiro-Wilk		
	membership	Statistic	df	Sig.	Statistic	df	Sig.
Exemplar 6	0	.155	29	.073	.968	29	.512
	1	.150	23	.194	.925	23	.084

a. Lilliefors Significance Correction

Test of Homogeneity of Variance

		Levene Statistic	df1	df2	Sig.
Exemplar 6	Based on Mean	19.440	1	50	.000

Figure 50. Re-test of assumptions for membership groups proximity to Exemplar 6 with outliers removed.

Results of Statistical Significance of *t*-tests

Eight independent samples *t*-tests were calculated to compare the mean scores of participants in collection 3 (membership = 0) to those of collection 13 (membership = 1) on the proximities to sorts 6, 15, 21, 84, 85, 185, 193, and 201 (Figure 51).

No statistically significant difference was found for participant sort proximities to sorts 15 [$t(36.9) = 1.66, p > .05$], 84 [$t(53) = -.58, p > .05$], 85 [$t(53) = -.08, p > .05$], or 193 [$t(33.7) = 1.70, p > .05$].

A statistically significant difference was found for participant sort proximities to sort 6 [$t(28.2) = 3.09, p = .004$]. The mean proximity for members of collection 3 ($m = 15.62, sd = 2.19$) was statistically significantly farther from sort 6 than the mean proximity for members of collection 13 ($m = 12.04, sd = 5.19$). An *r* of .25 was calculated which is a small to medium effect.

The sorts of participants in collection 13 were found to be closer in proximity to sort 6 than members of collection 3 and that this difference between groups accounted for 6% of the variance in proximity values.

A statistically significant difference was found for participant sort proximities to sort 21 [$t(31.1) = 2.47, p = .019$]. The mean proximity for members of collection 3 ($m = 14.50, sd = 2.08$) was statistically significantly further from sort 21 than the mean proximity for members of collection 13 ($m = 12.30, sd = 3.88$). An *r* of .16 was calculated which is a small effect.

The sorts of participants in collection 13 were found to be closer in proximity to sort 21 than members of collection 3 and that this difference between groups accounted for 3% of the variance in proximity values.

		Levene's Test				t-test for Equality of Means				
		F	Sig.	t	df	Sig. (2-tailed)	Mean Difference	Std. Error Difference	95% Confidence Interval of the Difference	
									Lower	Upper
Exemplar 6	Equal variances assumed	19.440	.000	3.358	50	.002	3.577	1.065	1.438	5.717
	Equal variances not assumed			3.093	28.231	.004	3.577	1.157	1.209	5.946
Exemplar 15	Equal variances assumed	4.662	.035	1.759	53	.084	1.466	.833	-.206	3.138
	Equal variances not assumed			1.661	36.944	.105	1.466	.883	-.322	3.254
Exemplar 21	Equal variances assumed	10.490	.002	2.712	53	.009	2.196	.810	.572	3.820
	Equal variances not assumed			2.472	31.087	.019	2.196	.888	.384	4.007
Exemplar 185	Equal variances assumed	1.426	.238	2.288	53	.026	1.338	.585	.165	2.511
	Equal variances not assumed			2.203	40.510	.033	1.338	.607	.111	2.566
Exemplar 193	Equal variances assumed	6.574	.013	1.839	53	.072	1.311	.713	-.119	2.741
	Equal variances not assumed			1.704	33.724	.098	1.311	.769	-.253	2.875
Exemplar 201	Equal variances assumed	3.987	.051	2.234	53	.030	2.534	1.134	.259	4.809
	Equal variances not assumed			2.130	38.714	.040	2.534	1.190	.127	4.941
Sort 84	Equal variances assumed	.908	.345	-.580	53	.564	-.257	.443	-1.145	.631
	Equal variances not assumed			-.592	50.590	.557	-.257	.434	-1.128	.615
Sort 85	Equal variances assumed	.244	.623	-.075	53	.941	-.034	.453	-.943	.875
	Equal variances not assumed			-.074	45.394	.941	-.034	.459	-.958	.890

Figure 51. Independent samples t-test between membership groups for proximities to each exemplar sort.

A statistically significant difference was found for participant sort proximities to sort 185 [$t(53) = 2.29$, $p = .026$]. The mean proximity for members of collection 3 ($m = 13.47$, $sd = 1.92$) was statistically significantly further from sort 185 than the mean

proximity for members of collection 13 ($m = 12.13$, $sd = 2.42$). An r of .09 was calculated which is a small effect.

The sorts of participants in collection 13 were found to be closer in proximity to sort 185 than members of collection 3 and that this difference between groups accounted for less than 1% of the variance in proximity values.

A statistically significant difference was found for participant sort proximities to sort 201 [$t(53) = 2.23$, $p = .03$]. The mean proximity for members of collection 3 ($m = 13.97$, $sd = 3.60$) was statistically significantly further from sort 201 than the mean proximity for members of collection 13 ($m = 11.43$, $sd = 4.82$). An r of .08 was calculated which is a small effect.

The sorts of participants in collection 13 were found to be closer in proximity to sort 201 than members of collection 3 and that this difference between groups accounted for less than 1% of the variance in proximity values.

Assumption Testing for Binary Logistic Regression (Mertler & Vannatta, 2013)

After removal of cases of multivariate and univariate outliers for the independent samples t -tests, there are 52 cases representing participants in collections 3 ($n = 29$) and 13 ($n = 23$). Testing with all four predictors at once yields a ratio of 13 cases to predictor. A minimum of 15 cases per predictor is recommended (Pallant, 2001). Therefore, Logistic regression will be performed on combinations of three of the four predictors at a time, yielding a ratio greater than 17.

Evidence of multicollinearity among predictors was investigated by performing a multiple regression and examining the Collinearity Statistics (Figure 52). Tolerances for

each of the predictors were greater than .1 and all of the VIFs were less than 10 indicating that no predictors explained the same construct.

Coefficients^a									
Model	Unstandardized Coefficients		Standardized Coefficients	t	Sig.	95.0% Confidence Interval for B		Collinearity Statistics	
	B	Std. Error	Beta			Lower Bound	Upper Bound	Tolerance	VIF
1 (Constant)	2.057	.401		5.135	.000	1.251	2.863		
Exemplar 6	-.058	.027	-.486	-2.146	.037	-.113	-.004	.281	3.557
Exemplar 21	-.059	.025	-.360	-2.307	.026	-.110	-.007	.593	1.685
Exemplar 201	.004	.027	.035	.151	.881	-.051	.059	.270	3.699
Exemplar 185	-.005	.036	-.022	-.136	.892	-.078	.068	.560	1.787

a. Dependent Variable: membership

Figure 52. Collinearity statistics for Exemplars 6, 21, 201, and 185.

Results of Logistic Regression

A forward binary logistic regression was performed (Figure 53) to determine which of the independent variables (proximities to exemplars 6, 21, 185, and 201) are reliable predictors of participant membership in collection 13, the group most likely to have conceptual representations that demonstrate attainment of the desired level of learning. Analysis was performed on combinations of three of the four predictors at a time as discussed above in the assumptions. Using the predictors 6, 21, and 201 produced the most favorable results with the overall model fit with the predictors proximity to exemplar 6, and proximity to exemplar 21 being statistically reliable in distinguishing between sorts submitted by members of collections 3 (membership = 0) and 13 (membership = 1) [-2 Log likelihood = 51.76, $X^2(2) = 19.63$, $p < .001$]. The model

correctly classified 15 of 23 (65.2%) as being members in collection 13, and correctly classified 73.1 % of cases overall.

Wald statistics (*Figure 54*) indicate that the proximity of a student's sorts to both exemplar 6 and exemplar 21 significantly predicted their level of attainment of student outcome 2. The odds ratios for these two variables indicate a decrease in the likelihood of having attained the learning objective as the proximity value to either of these exemplars increases, or stated conversely, a closer proximity to the exemplars increases the likelihood of a student demonstrating a level of conceptual development representative of having attained the learning objective.

Model Summary

Step	-2 Log likelihood	Cox & Snell R Square	Nagelkerke R Square
1	60.703 ^a	.186	.249
2	51.762 ^a	.314	.421

a. Estimation terminated at iteration number 5 because parameter estimates changed by less than .001.

Omnibus Tests of Model Coefficients

		Chi-square	df	Sig.
Step 1	Step	10.690	1	.001
	Block	10.690	1	.001
	Model	10.690	1	.001
Step 2	Step	8.941	1	.003
	Block	19.631	2	.000
	Model	19.631	2	.000

Classification Table^a

			Predicted		Percentage Correct
Observed			membership 0	1	
Step 1	membership	0	24	5	82.8
		1	10	13	56.5
	Overall Percentage				71.2
Step 2	membership	0	23	6	79.3
		1	8	15	65.2
	Overall Percentage				73.1

a. The cut value is .500

Figure 53. Logistic Regression Model results showing correct prediction rate for membership in the most-likely with two variables is 65%.

		Variables in the Equation					
		B	S.E.	Wald	df	Sig.	Exp(B)
Step 1 ^a	Exemplar 6	-.269	.100	7.262	1	.007	.764
	Constant	3.582	1.479	5.865	1	.015	35.949
Step 2 ^b	Exemplar 21	-.358	.140	6.568	1	.010	.699
	Exemplar 6	-.328	.113	8.399	1	.004	.720
	Constant	9.259	2.945	9.887	1	.002	10502.616

a. Variable(s) entered on step 1: Exemplar 6.

b. Variable(s) entered on step 2: Exemplar 21.

Figure 54. Logistic Regression results showing Exemplars 6 and 21 in the solution.

APPENDIX E

Statistical Analysis for Question 3

Research Question 3

Is there a statistically significant difference among the categories of computer science students' progression through milestones of coursework attainment (introductory, completing, and mid-program) on the dependent variable, the edit distance between participant's card sorts to the exemplar sorts (sorts 6, 21, 185, and 201)?

Null and Alternative Hypotheses

H_0 = There are no statistically significant differences among the categories of computer science students' progression through milestones of coursework attainment (introductory, completing, and mid-program) on the dependent variable, the edit distance between participant's card sorts to the exemplar sorts (sorts 6, 21, 185, and 201).

H_a = There are statistically significant differences among the categories of computer science students' progression through milestones of coursework attainment (introductory, completing, and mid-program) on the dependent variable, the edit distance between participant's card sorts to the exemplar sorts (sorts 6, 21, 185, and 201).

Assumption Testing for ANOVA

There are no missing data values for testing Proximity to the Exemplar sorts by Coursework Attainment (Figure 55). It is noted that the categorizations of participants by levels of coursework attainment result in three groups of unequal sizes (i.e., $n = 50, 45,$ and 29).

The groups are independent as the membership in the categorical groups of coursework attainment is mutually exclusive. The dependent variables are the scalar edit distance between participant sorts and each exemplar sort.

Case Processing Summary

		Cases					
		Valid		Missing		Total	
	Coursework Attainment	N	Percent	N	Percent	N	Percent
Proximity to 6	Introductory	50	100.0%	0	0.0%	50	100.0%
	Mid Program	45	100.0%	0	0.0%	45	100.0%
	Completing	29	100.0%	0	0.0%	29	100.0%
Proximity to 21	Introductory	50	100.0%	0	0.0%	50	100.0%
	Mid Program	45	100.0%	0	0.0%	45	100.0%
	Completing	29	100.0%	0	0.0%	29	100.0%
Proximity to 185	Introductory	50	100.0%	0	0.0%	50	100.0%
	Mid Program	45	100.0%	0	0.0%	45	100.0%
	Completing	29	100.0%	0	0.0%	29	100.0%
Proximity to 201	Introductory	50	100.0%	0	0.0%	50	100.0%
	Mid Program	45	100.0%	0	0.0%	45	100.0%
	Completing	29	100.0%	0	0.0%	29	100.0%

Figure 55. Case Process Summary for mean values of proximity to Exemplars 6, 21, 185, and 201 for categories of coursework attainment (Introductory, Mid program, Completing).

Outliers were found in the several groups of data as shown in Figure 56. Three cases were found among the Introductory students' sorts proximity to Exemplar 6. These data points fell below two standard deviations from the mean. These data were recoded, in a new variable, to the lowest value (i.e., 7) within two standard deviations from the mean. Also, the exemplar sorts 21 and 185 were submitted by participants in the Completing group. This resulted in outlying values of 0 for testing the proximity to Exemplar 21, and to Exemplar 185. These data were recoded, in new variables, to the

lowest value within two standard deviations from the mean: 5 for proximity to Exemplar 21, and 7 for proximity to Exemplar 185.

After recoding the outliers, all data fall within the box and whiskers plots and are included in the following descriptive statistics and analyses.

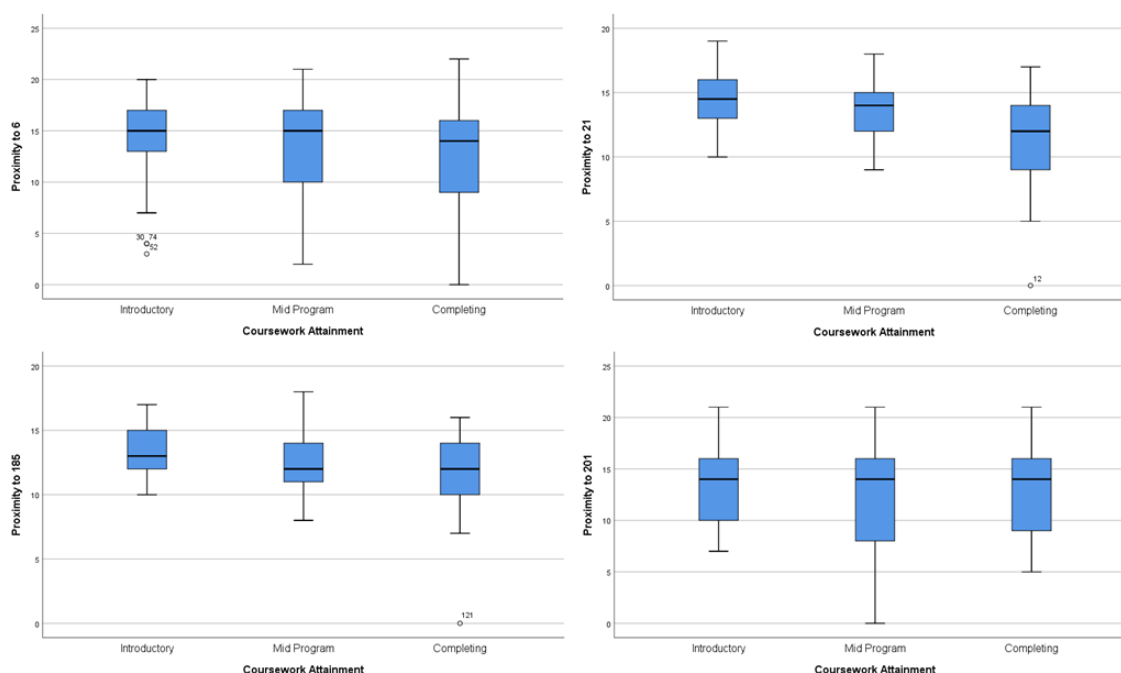


Figure 56. Box and whiskers plots for of proximity to Exemplars 6, 21, 185, and 201 for categories of coursework attainment (Introductory, Mid program, Completing).

An inspection of the Kolmogorov-Smirnov (Figure 57) indicated significance levels lower than .05 for all groups except the Completing students in proximity to Exemplars 6 and 185. Calculation of the z-scores (Figure 58) indicates that all data values fall within ± 3 standard deviations. Therefore, normality is assumed.

Levene's test for homogeneity was found to be significant, indicating homogeneity, when comparing groups for Proximity to Exemplars 185 and 201, but not significant when comparing groups for Proximity to Exemplars 6 or 21.

Tests of Normality

		Kolmogorov-Smirnov ^a			Shapiro-Wilk		
Coursework Attainment		Statistic	df	Sig.	Statistic	df	Sig.
Proximity to 6	Introductory	.130	50	.035	.940	50	.014
	Mid Program	.177	45	.001	.932	45	.011
	Completing	.152	29	.084	.970	29	.546
Proximity to 21	Introductory	.137	50	.019	.961	50	.096
	Mid Program	.151	45	.012	.961	45	.136
	Completing	.186	29	.012	.946	29	.148
Proximity to 185	Introductory	.126	50	.045	.946	50	.023
	Mid Program	.158	45	.007	.956	45	.088
	Completing	.152	29	.086	.955	29	.241
Proximity to 201	Introductory	.136	50	.022	.952	50	.042
	Mid Program	.158	45	.007	.951	45	.056
	Completing	.167	29	.037	.946	29	.141

a. Lilliefors Significance Correction

Figure 57. Test of Normality of proximity to Exemplars 6, 21, 185, and 201 as categorized by coursework attainment (Introductory, Mid program, Completing).

Z-Scores

Proximity to Exemplar:	Category	Skewness		Kurtosis		z-Scores	
		Statistic	Std Err	Statistic	Std Err	Skewness	Kurtosis
6	Intro	-0.583	0.337	-0.301	0.662	-1.730	-0.455
	Mid	-0.545	0.354	-0.835	0.695	-1.540	-1.201
	Completing	-0.491	0.434	-0.124	0.845	-1.131	-0.147
21	Intro	-0.088	0.337	-0.509	0.662	-0.261	-0.769
	Mid	-0.247	0.354	-0.532	0.695	-0.698	-0.765
	Completing	-0.419	0.434	-0.622	0.845	-0.965	-0.736
185	Intro	0.008	0.337	-1.001	0.662	0.024	-1.512
	Mid	0.381	0.354	0.202	0.695	1.076	0.291
	Completing	-0.389	0.434	-0.47	0.845	-0.896	-0.556
201	Intro	-0.069	0.337	-1.05	0.662	-0.205	-1.586
	Mid	-0.422	0.354	-0.459	0.695	-1.192	-0.660
	Completing	-0.168	0.434	-1.093	0.845	-0.387	-1.293

Figure 58. Z-Scores for proximities to Exemplars 6, 21, 185, and 201 as categorized by coursework attainment (Introductory, Mid program, Completing).

Test of Homogeneity of Variance

		Levene Statistic	df1	df2	Sig.
Proximity to 6	Based on Mean	5.618	2	121	.005
	Based on Median	3.149	2	121	.046
	Based on Median and with adjusted df	3.149	2	108.487	.047
	Based on trimmed mean	5.305	2	121	.006
Proximity to 21	Based on Mean	5.314	2	121	.006
	Based on Median	3.466	2	121	.034
	Based on Median and with adjusted df	3.466	2	93.967	.035
	Based on trimmed mean	5.139	2	121	.007
Proximity to 185	Based on Mean	.567	2	121	.569
	Based on Median	.654	2	121	.522
	Based on Median and with adjusted df	.654	2	114.627	.522
	Based on trimmed mean	.567	2	121	.569
Proximity to 201	Based on Mean	1.822	2	121	.166
	Based on Median	1.005	2	121	.369
	Based on Median and with adjusted df	1.005	2	107.823	.370
	Based on trimmed mean	1.777	2	121	.174

Figure 59. Test of Homogeneity of Variance for proximities to Exemplars 6, 21, 185, and 201 as categorized by coursework attainment (Introductory, Mid program, Completing).

Summary of Assumption Testing for ANOVA: The assumptions of dependent Interval data and Independence of the factoring variables are met for proximity to the four exemplar sorts of Research Question 3. Outlying data values, including two of the exemplar sorts themselves, were recoded to values two standard deviations below the mean. Based upon z-scores, normality was assumed. Homogeneity of variance among groups was indicated for comparing proximities to exemplars 185 and 201 so ANOVA post hoc analysis will be performed using Bonferroni. Since unequal variances were indicated among the groups for comparison of proximities to exemplars 6 and 21, ANOVA post hoc analysis will be performed using Dunnett T3, which yields conservative results with unequal variances and equal or unequal group sizes (Shingala & Rajyaguru, 2015).

Results of Statistical Significance for Proximity to Exemplar Sort 6

A one-way ANOVA compared participants' sort proximity to exemplar sort 6 grouped by participant coursework attainment (Figure 60). For the entire model, no statistically significant difference was found based on category of coursework attainment, $F(2,121) = 1.15, p > .05, \eta^2 = .02$. The effect size was $\eta^2 = .02$; a small effect size (Kirk, 1996) indicating that 2% of the variance of proximities to exemplar sort 6 was explained by a student's level of coursework attainment.

Tests of Between-Subjects Effects

Dependent Variable: Proximity to 6

Source	Type III Sum of Squares	df	Mean Square	F	Sig.	Partial Eta Squared
Corrected Model	47.220 ^a	2	23.610	1.148	.321	.019
Intercept	21248.796	1	21248.796	1033.068	.000	.895
courseCategory	47.220	2	23.610	1.148	.321	.019
Error	2488.805	121	20.569			
Total	25433.000	124				
Corrected Total	2536.024	123				

a. R Squared = .019 (Adjusted R Squared = .002)

Figure 60. One-way ANOVA comparing proximities to Exemplar sort 6 by category of coursework attainment.

The analysis revealed the following means and standard deviations (Figure 61) for each level of coursework attainment: a) Introductory students ($m = 14.26, sd = 3.52$), b) Mid program students ($m = 13.42, sd = 5.09$), and c) Completing students ($m = 12.69, sd = 5.15$). Post hoc tests using Dunnett T3 found no statistically significant difference among the groups (Figure 62).

Descriptive Statistics

Dependent Variable: Proximity to 6

Coursework Attainment	Mean	Std. Deviation	N
Introductory	14.26	3.516	50
Mid Program	13.42	5.092	45
Completing	12.69	5.149	29
Total	13.59	4.541	124

Figure 61. Mean and standard deviation statistics of proximities to Exemplar 6 for the Introductory, Mid program, and Completing groups of participants.

Multiple Comparisons

Dependent Variable: Proximity to 6

Dunnett T3

(I) Coursework Attainment	(J) Coursework Attainment	Mean Difference (I-J)	Std. Error	Sig.	95% Confidence Interval	
					Lower Bound	Upper Bound
Introductory	Mid Program	.84	.907	.733	-1.38	3.05
	Completing	1.57	1.078	.386	-1.10	4.24
Mid Program	Introductory	-.84	.907	.733	-3.05	1.38
	Completing	.73	1.221	.908	-2.26	3.73
Completing	Introductory	-1.57	1.078	.386	-4.24	1.10
	Mid Program	-.73	1.221	.908	-3.73	2.26

Based on observed means.

The error term is Mean Square(Error) = 20.569.

Figure 62. Post hoc pair-wise comparisons among the categories of coursework attainment for proximity to Exemplar 6.

Note in Figure 63 that higher proximity values indicate that a participant's sort has low similarity with the exemplar; while lower proximity values indicate the participant sort is more similar to the exemplar. Examination of the mean proximities for each level of coursework attainment indicate that proximity to the exemplar decreased, becoming more similar to the exemplar, relative to progressive levels of coursework attainment (introductory, mid-program, and completing). Introductory student sorts had

the furthest proximity from the exemplar while completing student sorts had the nearest proximities to the exemplar.

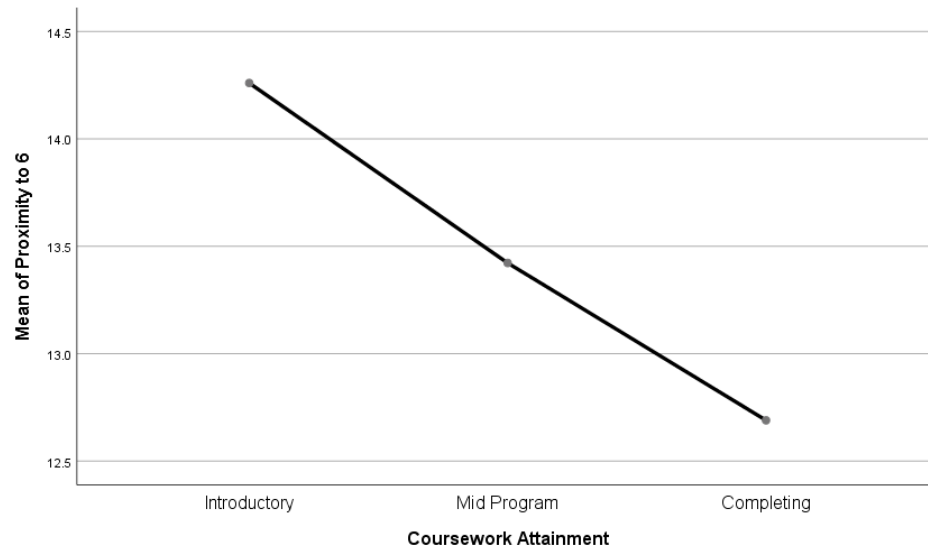


Figure 63. Line chart of mean proximity values to Exemplar 6 by categories of coursework attainment.

Results of Statistical Significance for Proximity to Exemplar Sort 21

A one-way ANOVA compared participants' sort proximity to exemplar sort 21 grouped by participant coursework attainment (Figure 64). For the entire model, a statistically significant difference was found based on category of coursework attainment, $F(2,121) = 13.32, p < .001, \eta^2 = .18$. The effect size was $\eta^2 = .18$; a large effect size (Kirk, 1996) indicating that 18% of the variance of proximities to exemplar sort 21 was explained by a student's level of coursework attainment.

Tests of Between-Subjects Effects

Dependent Variable: Proximity to 21

Source	Type III Sum of Squares	df	Mean Square	F	Sig.	Partial Eta Squared
Corrected Model	167.914 ^a	2	83.957	13.322	.000	.180
Intercept	20326.560	1	20326.560	3225.295	.000	.964
courseCategory	167.914	2	83.957	13.322	.000	.180
Error	762.570	121	6.302			
Total	23314.000	124				
Corrected Total	930.484	123				

a. R Squared = .180 (Adjusted R Squared = .167)

Figure 64. One-way ANOVA comparing proximities to Exemplar sort 21 by category of coursework attainment.

The analysis revealed the following means and standard deviations (Figure 65) for each level of coursework attainment: a) Introductory students ($m = 14.46$, $sd = 2.09$), b) Mid program students ($m = 13.58$, $sd = 2.29$), and c) Completing students ($m = 11.45$, $sd = 3.37$). Post hoc tests using Dunnett T3 found a statistically significant difference in the means of Completing students to Mid program students ($p = .013$), and to Introductory students ($p < .001$) respectively (Figure 66).

Descriptive Statistics

Dependent Variable: Proximity to 21

Coursework Attainment	Mean	Std. Deviation	N
Introductory	14.46	2.092	50
Mid Program	13.58	2.291	45
Completing	11.45	3.366	29
Total	13.44	2.750	124

Figure 65. Mean and standard deviation statistics of proximities to Exemplar 21 for the Introductory, Mid program, and Completing groups of participants.

Multiple Comparisons

Dependent Variable: Proximity to 21

Dunnett T3

(I) Coursework Attainment	(J) Coursework Attainment	Mean Difference (I-J)	Std. Error	Sig.	95% Confidence Interval	
					Lower Bound	Upper Bound
Introductory	Mid Program	.88	.452	.153	-.22	1.98
	Completing	3.01*	.691	.000	1.29	4.73
Mid Program	Introductory	-.88	.452	.153	-1.98	.22
	Completing	2.13*	.712	.013	.37	3.89
Completing	Introductory	-3.01*	.691	.000	-4.73	-1.29
	Mid Program	-2.13*	.712	.013	-3.89	-.37

Based on observed means.

The error term is Mean Square(Error) = 6.302.

*. The mean difference is significant at the .05 level.

Figure 66. Post hoc pair-wise comparisons among the categories of coursework attainment for proximity to Exemplar 21.

Note in *Figure 67* that higher proximity values indicate that a participant's sort has low similarity with the exemplar; while lower proximity values indicate the participant sort is more similar to the exemplar. Examination of the mean proximities for each level of coursework attainment indicate that proximity to the exemplar decreased, becoming more similar to the exemplar, relative to progressive levels of coursework attainment (introductory, mid-program, and completing). Introductory student sorts had the furthest proximity from the exemplar while completing student sorts had the nearest proximities to the exemplar.

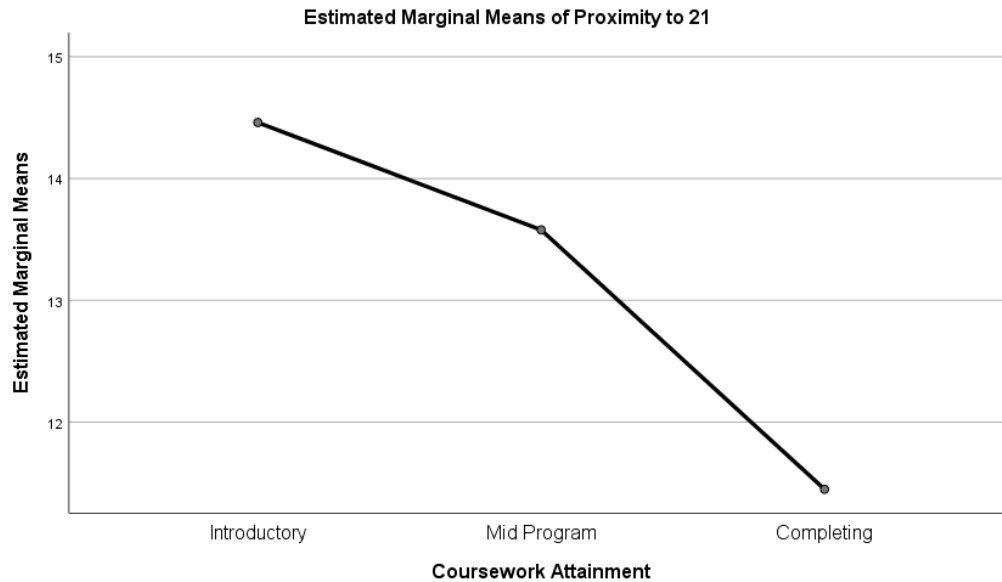


Figure 67. Line chart of mean proximity values to Exemplar 21 by categories of coursework attainment.

Results of Statistical Significance for Proximity to Exemplar Sort 185

A one-way ANOVA compared participants' sort proximity to exemplar sort 185 grouped by participant coursework attainment (Figure 68). For the entire model, a statistically significant difference was found based on category of attainment, $F(2,121) = 3.35$, $p = .038$, $\eta^2 = .05$. The effect size was $\eta^2 = .05$; a small effect size (Kirk, 1996) indicating that 5% of the variance of proximities to exemplar sort 185 was explained by a student's level of coursework attainment.

Tests of Between-Subjects Effects

Dependent Variable: Proximity to 185

Source	Type III Sum of Squares	df	Mean Square	F	Sig.	Partial Eta Squared
Corrected Model	31.158 ^a	2	15.579	3.347	.038	.052
Intercept	18589.388	1	18589.388	3993.838	.000	.971
courseCategory	31.158	2	15.579	3.347	.038	.052
Error	563.197	121	4.655			
Total	20574.000	124				
Corrected Total	594.355	123				

a. R Squared = .052 (Adjusted R Squared = .037)

Figure 68. One-way ANOVA comparing proximities to Exemplar sort 185 by category of coursework attainment.

The analysis revealed the following means and standard deviations (Figure 69) for each level of coursework attainment: a Introductory students ($m = 13.24$, $sd = 2.04$), b) Mid program students ($m = 12.56$, $sd = 2.15$), and c) Completing students ($m = 11.97$, $sd = 2.37$). Post hoc tests using Bonferroni found a statistically significant difference in the means of Completing students to Introductory students ($p = .038$) as shown in Figure 70.

Descriptive Statistics

Dependent Variable: Proximity to 185

Coursework Attainment	Mean	Std. Deviation	N
Introductory	13.24	2.036	50
Mid Program	12.56	2.149	45
Completing	11.97	2.368	29
Total	12.69	2.198	124

Figure 69. Mean and standard deviation statistics of proximities to Exemplar 185 for the Introductory, Mid program, and Completing groups of participants.

Multiple Comparisons

Dependent Variable: Proximity to 185

Bonferroni

(I) Coursework Attainment	(J) Coursework Attainment	Mean Difference (I-J)	Std. Error	Sig.	95% Confidence Interval	
					Lower Bound	Upper Bound
Introductory	Mid Program	.68	.443	.376	-.39	1.76
	Completing	1.27 [*]	.504	.038	.05	2.50
Mid Program	Introductory	-.68	.443	.376	-1.76	.39
	Completing	.59	.514	.759	-.66	1.84
Completing	Introductory	-1.27 [*]	.504	.038	-2.50	-.05
	Mid Program	-.59	.514	.759	-1.84	.66

Based on observed means.

The error term is Mean Square(Error) = 4.655.

*. The mean difference is significant at the .05 level.

Figure 70. Post hoc pair-wise comparisons among the categories of coursework attainment for proximity to Exemplar 185.

Note in Figure 71 that higher proximity values indicate that a participant's sort has low similarity with the exemplar; while lower proximity values indicate the participant sort is more similar to the exemplar. Examination of the mean proximities for each level of coursework attainment indicate that proximity to the exemplar decreased, becoming more similar to the exemplar, relative to progressive levels of coursework attainment (introductory, mid-program, and completing). Introductory student sorts had the furthest proximity from the exemplar while completing student sorts had the nearest proximities to the exemplar.

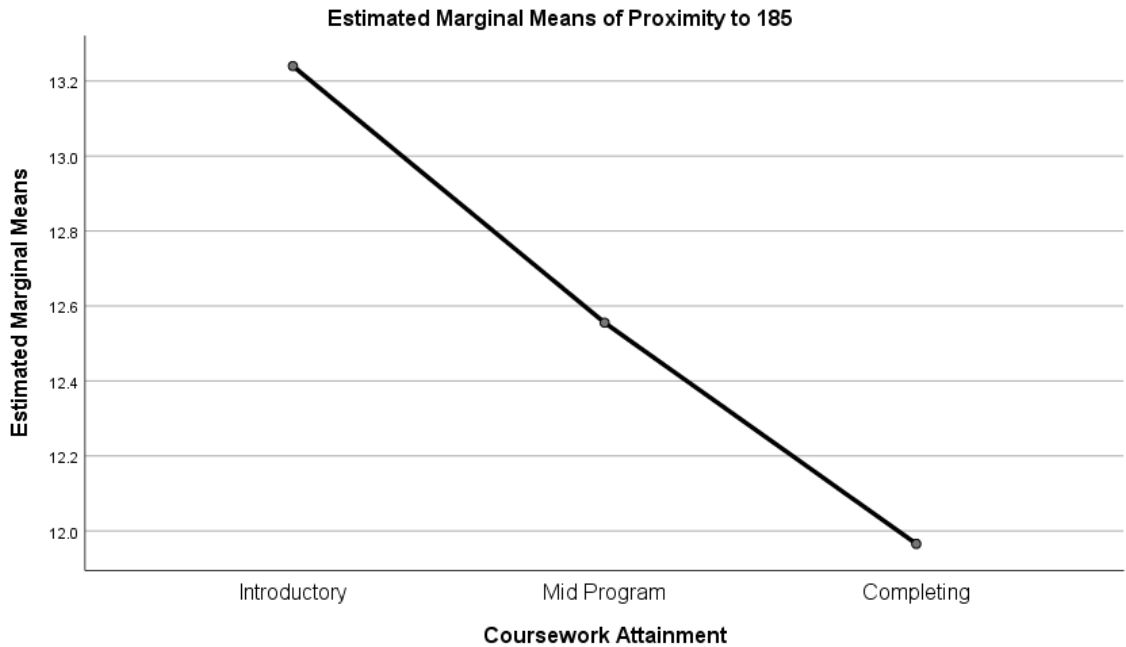


Figure 71. Line chart of mean proximity values to Exemplar 185 by categories of coursework attainment.

Results of Statistical Significance for Proximity to Exemplar Sort 201

A one-way ANOVA compared participants' sort proximity to exemplar sort 201 grouped by participant coursework attainment (Figure 72). For the entire model, no statistically significant difference was found based on category of coursework attainment, $F(2,121) = .641, p > .05, \eta^2 = .01$. The effect size was $\eta^2 = .01$; a small effect size (Kirk, 1996) indicating that 1% of the variance of proximities to exemplar sort 201 was explained by a student's level of coursework attainment.

Tests of Between-Subjects Effects

Dependent Variable: Proximity to 201

Source	Type III Sum of Squares	df	Mean Square	F	Sig.	Partial Eta Squared
Corrected Model	24.315 ^a	2	12.158	.641	.528	.010
Intercept	19587.933	1	19587.933	1032.973	.000	.895
courseCategory	24.315	2	12.158	.641	.528	.010
Error	2294.483	121	18.963			
Total	23145.000	124				
Corrected Total	2318.798	123				

a. R Squared = .010 (Adjusted R Squared = -.006)

Figure 72. One-way ANOVA comparing proximities to Exemplar sort 201 by category of coursework attainment.

The analysis revealed the following means and standard deviations (Figure 73) for each level of coursework attainment: a) Introductory students ($m = 13.48$, $sd = 3.80$), b) Mid program students ($m = 12.49$, $sd = 4.76$), and c) Completing students ($m = 12.79$, $sd = 4.59$). Post hoc tests using Bonferroni found no statistically significant difference among the groups (Figure 74).

Descriptive Statistics

Dependent Variable: Proximity to 201

Coursework Attainment	Mean	Std. Deviation	N
Introductory	13.48	3.797	50
Mid Program	12.49	4.761	45
Completing	12.79	4.593	29
Total	12.96	4.342	124

Figure 73. Mean and standard deviation statistics of proximities to Exemplar 201 for the Introductory, Mid program, and Completing groups of participants.

Multiple Comparisons

Dependent Variable: Proximity to 201

Bonferroni

(I) Coursework Attainment	(J) Coursework Attainment	Mean Difference (I-J)	Std. Error	Sig.	95% Confidence Interval	
					Lower Bound	Upper Bound
Introductory	Mid Program	.99	.895	.811	-1.18	3.16
	Completing	.69	1.016	1.000	-1.78	3.15
Mid Program	Introductory	-.99	.895	.811	-3.16	1.18
	Completing	-.30	1.037	1.000	-2.82	2.21
Completing	Introductory	-.69	1.016	1.000	-3.15	1.78
	Mid Program	.30	1.037	1.000	-2.21	2.82

Based on observed means.

The error term is Mean Square(Error) = 18.963.

Figure 74. Post hoc pair-wise comparisons among the categories of coursework attainment for proximity to Exemplar 201.

Note in Figure 75 that higher proximity values indicate that a participant's sort has low similarity with the exemplar; while lower proximity values indicate the participant sort is more similar to the exemplar. Examination of the mean proximities for each level of coursework attainment indicate that proximity to the exemplar decreased, becoming more similar to the exemplar, as students progressed from the Introductory to Mid program level of coursework and then increased for Completing students. Introductory student sorts had the furthest proximity from the exemplar while mid program student sorts had the nearest proximities to the exemplar.

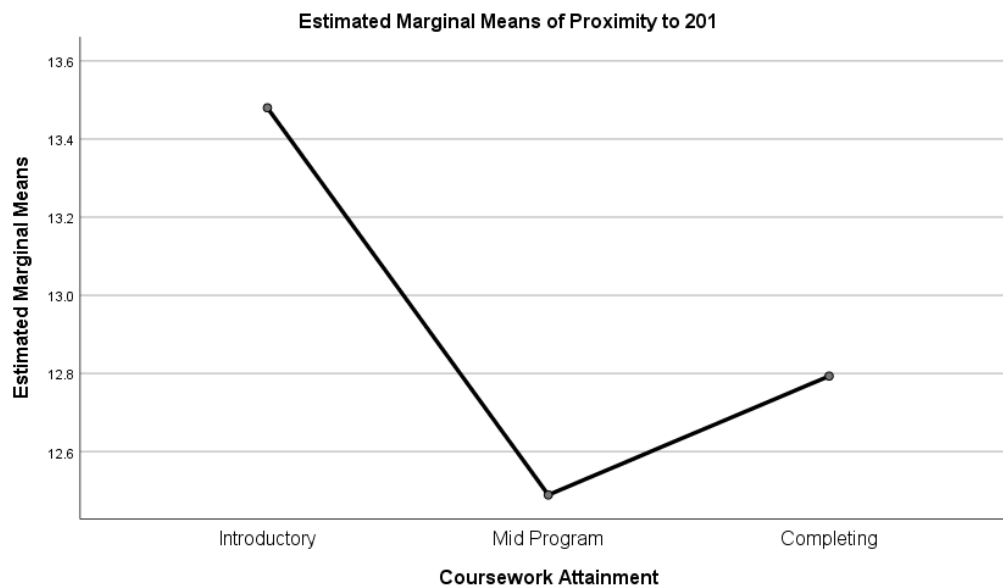


Figure 75. Line chart of mean proximity values to Exemplar 201 by categories of coursework attainment.

APPENDIX F

Statistical Analysis for Question 4

Research Question 4

Is there a statistically significant difference among the categories of computer science students' purposeful programming experience (light, moderate, and extensive) on the dependent variable, the edit distance between participant's card sorts to the exemplar sorts (sorts 6, 21, 185, and 201)?

Null and Alternative Hypotheses

H_0 = There are no statistically significant differences among the categories of computer science students' purposeful programming experience (light, moderate, and extensive) on the dependent variable, the edit distance between participant's card sorts to the exemplar sorts (sorts 6, 21, 185, and 201).

H_a = There are statistically significant differences among the categories of computer science students' purposeful programming experience (light, moderate, and extensive) on the dependent variable, the edit distance between participant's card sorts to the exemplar sorts (sorts 6, 21, 185, and 201).

Assumption Testing for ANOVA

There are no missing data values for testing Proximity to the Exemplar sorts by Purposeful Experience (Figure 76). It is noted that the Light and Extensive groups are of equal size while the Moderate group is 10% smaller.

The groups are independent as the membership in the categorical groups of purposeful programming experience is mutually exclusive. The dependent variables are the scalar edit distance between participant sorts and each exemplar sort.

Case Processing Summary

		Cases					
		Valid		Missing		Total	
Purposeful Experience	N	Percent	N	Percent	N	Percent	
Proximity to 6	Light	43	100.0%	0	0.0%	43	100.0%
	Moderate	38	100.0%	0	0.0%	38	100.0%
	Extensive	43	100.0%	0	0.0%	43	100.0%
Proximity to 21	Light	43	100.0%	0	0.0%	43	100.0%
	Moderate	38	100.0%	0	0.0%	38	100.0%
	Extensive	43	100.0%	0	0.0%	43	100.0%
Proximity to 185	Light	43	100.0%	0	0.0%	43	100.0%
	Moderate	38	100.0%	0	0.0%	38	100.0%
	Extensive	43	100.0%	0	0.0%	43	100.0%
Proximity to 201	Light	43	100.0%	0	0.0%	43	100.0%
	Moderate	38	100.0%	0	0.0%	38	100.0%
	Extensive	43	100.0%	0	0.0%	43	100.0%

Figure 76. Case Process Summary for mean values of proximity to Exemplars 6, 21, 185, and 201 for categories of purposeful experience (Light, Moderate, Extensive).

Outliers were found in the several groups of data as shown in Figure 77. Three cases were found among moderately experienced students' sorts proximity to Exemplar 6. These data points fell below two standard deviations from the mean. These data were recoded, in a new variable, to the lowest value (i.e., 7) within two standard deviations from the mean. Also, the exemplar sorts 21 and 185 resulted in outlying values of 0 for light experience students in testing the proximity to Exemplar 21, and to Exemplar 185. These data were recoded, in new variables, to the lowest value within two standard deviations from the mean: 5 for proximity to Exemplar 21, and 8 for proximity to Exemplar 185. One other outlier was found for students with extensive experience proximity to exemplar 21. This data point fell below two standard deviations from the mean. The data were recoded, in a new variable, to the lowest value (i.e., 11) within two standard deviations from the mean.

After recoding the outliers, all data fall within the box and whiskers plots and are included in the following descriptive statistics and analyses.

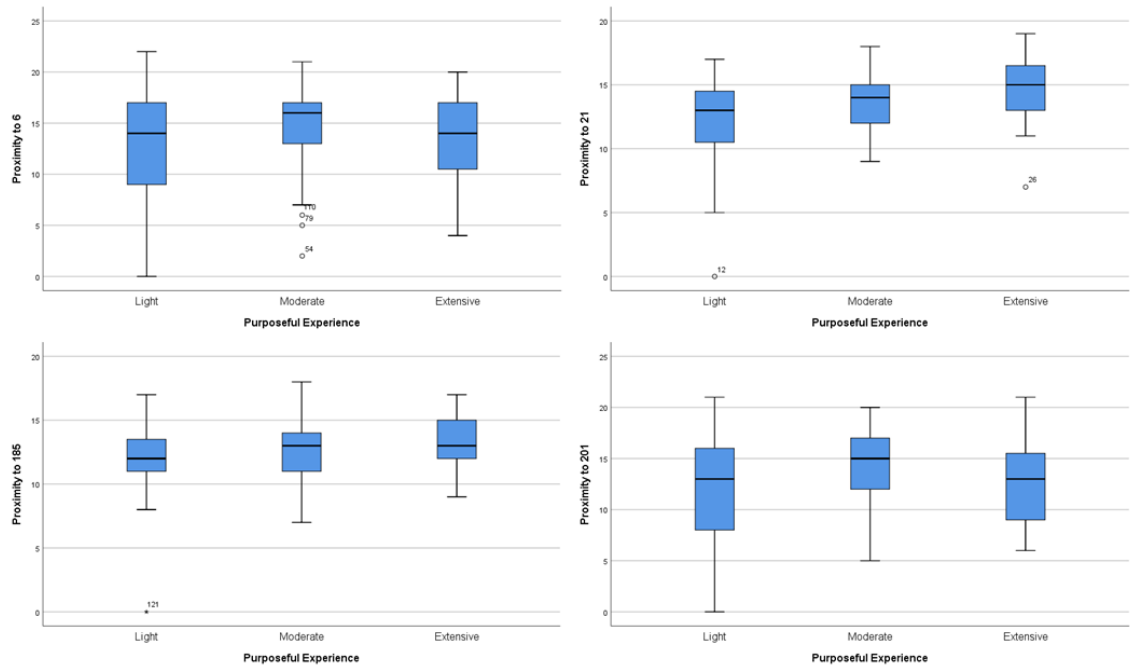


Figure 77. Box and Whiskers plots for proximity to Exemplars 6, 21, 185, and 201 for categories of purposeful experience (Light, Moderate, Extensive).

An inspection of the Kolmogorov-Smirnov (*Figure 78*) indicates significance levels lower than .05 for all groups except the Extensive category in proximity to Exemplar 6. Calculation of the z-scores (*Figure 79*) indicates that all data values fall within ± 3 standard deviations. Therefore, normality is assumed.

Levene's test for homogeneity was found to be significant, indicating homogeneity, when comparing groups for Proximity to Exemplars 21 and 185, but not significant when comparing groups for Proximity to Exemplars 6 or 201.

Tests of Normality

		Kolmogorov-Smirnov ^a			Shapiro-Wilk		
	Purposeful Experience	Statistic	df	Sig.	Statistic	df	Sig.
Proximity to 6	Light	.154	43	.012	.960	43	.143
	Moderate	.200	38	.001	.893	38	.002
	Extensive	.114	43	.190	.955	43	.094
Proximity to 21	Light	.184	43	.001	.935	43	.017
	Moderate	.175	38	.005	.947	38	.074
	Extensive	.143	43	.027	.945	43	.039
Proximity to 185	Light	.135	43	.047	.970	43	.306
	Moderate	.147	38	.038	.974	38	.514
	Extensive	.142	43	.029	.960	43	.137
Proximity to 201	Light	.148	43	.019	.963	43	.172
	Moderate	.170	38	.007	.924	38	.013
	Extensive	.158	43	.009	.943	43	.034

a. Lilliefors Significance Correction

Figure 78. Test of Normality for proximity to Exemplars 6, 21, 185, and 201 for categories of purposeful experience (Light, Moderate, Extensive)

Z-Scores

Proximity to Exemplar:		Skewness		Kurtosis		z-Scores	
	Category	Statistic	Std Err	Statistic	Std Err	Skewness	Kurtosis
6	Light	-0.46	0.361	-0.596	0.709	-1.274	-0.841
	Moderate	-0.85	0.383	0.023	0.75	-2.219	0.031
	Extensive	-0.476	0.361	-0.529	0.709	-1.319	-0.746
21	Light	-0.745	0.361	0.095	0.709	-2.064	0.134
	Moderate	-0.241	0.383	-0.35	0.75	-0.629	-0.467
	Extensive	-0.009	0.361	-0.964	0.709	-0.025	-1.360
185	Light	0.213	0.361	-0.276	0.709	0.590	-0.389
	Moderate	-0.114	0.383	-0.059	0.75	-0.298	-0.079
	Extensive	0.034	0.361	0.361	0.709	0.094	0.509
201	Light	-0.713	0.361	0.709	0.709	-1.975	1.000
	Moderate	-0.231	0.383	0.361	0.75	-0.603	0.481
	Extensive	-0.584	0.361	0.709	0.709	-1.618	1.000

Figure 79. Z-Scores for proximity to Exemplars 6, 21, 185, and 201 for categories of purposeful experience (Light, Moderate, Extensive).

Test of Homogeneity of Variance

		Levene Statistic	df1	df2	Sig.
Proximity to 6	Based on Mean	3.943	2	121	.022
	Based on Median	3.038	2	121	.052
	Based on Median and with adjusted df	3.038	2	113.464	.052
	Based on trimmed mean	3.795	2	121	.025
Proximity to 21	Based on Mean	2.318	2	121	.103
	Based on Median	1.327	2	121	.269
	Based on Median and with adjusted df	1.327	2	95.826	.270
	Based on trimmed mean	2.023	2	121	.137
Proximity to 185	Based on Mean	.369	2	121	.692
	Based on Median	.240	2	121	.787
	Based on Median and with adjusted df	.240	2	117.696	.787
	Based on trimmed mean	.362	2	121	.697
Proximity to 201	Based on Mean	3.378	2	121	.037
	Based on Median	3.260	2	121	.042
	Based on Median and with adjusted df	3.260	2	114.691	.042
	Based on trimmed mean	3.459	2	121	.035

Figure 80. Test of Homogeneity of Variance for proximity to Exemplars 6, 21, 185, and 201 for categories of purposeful experience (Light, Moderate, Extensive).

Summary of Assumption Testing for ANOVA: The assumptions of dependent Interval data and Independence of the factoring variables are met for proximity to the four exemplar sorts of Research Question 4. Outlying data values, including two of the exemplar sorts themselves, were recoded to values two standard deviations below the mean. Based upon z-scores, normality was assumed. Homogeneity of variance among groups was indicated for comparing proximities to exemplars 21 and 185 so ANOVA post hoc analysis will be performed using Bonferroni. Since unequal variances were indicated among the groups for comparison of proximities to exemplars 6 and 201, ANOVA post hoc analysis will be performed using Dunnett T3, which yields conservative results with unequal variances and equal or unequal group sizes (Shingala & Rajyaguru, 2015).

Results Statistical Significance for Proximity to Exemplar Sort 6

A one-way ANOVA compared participants' sort proximity to exemplar sort 6 grouped by level of participant programming experience (Figure 81). For the entire model, no statistically significant difference was found based on category of purposeful programming experience, $F(2,121) = 1.78$, $p > .05$, $\eta^2 = .03$. The effect size was $\eta^2 = .03$; a small effect size (Kirk, 1996) indicating that 3% of the variance of proximities to exemplar sort 6 was explained by a student's level of programming experience.

Tests of Between-Subjects Effects

Dependent Variable: Proximity to 6

Source	Type III Sum of Squares	df	Mean Square	F	Sig.	Partial Eta Squared
Corrected Model	73.500 ^a	2	36.750	1.784	.172	.029
Intercept	22899.722	1	22899.722	1111.527	.000	.902
practiceCategory	73.500	2	36.750	1.784	.172	.029
Error	2492.846	121	20.602			
Total	25409.000	124				
Corrected Total	2566.347	123				

a. R Squared = .029 (Adjusted R Squared = .013)

Figure 81. One-way ANOVA comparing proximities to Exemplar sort 6 by category of purposeful experience.

The analysis revealed the following means and standard deviations (Figure 82) for each level of programming experience: a) Light ($m = 12.70$, $sd = 5.34$), b) Moderate ($m = 14.61$, $sd = 3.70$), and c) Extensive ($m = 12.69$, $sd = 5.15$). Post hoc tests using Dunnett T3 found no statistically significant difference among the groups (Figure 83).

Descriptive Statistics

Dependent Variable: Proximity to 6

Purposeful Experience	Mean	Std. Deviation	N
Light	12.70	5.343	43
Moderate	14.61	3.702	38
Extensive	13.53	4.328	43
Total	13.57	4.568	124

Figure 82. Mean and standard deviation statistics of proximities to Exemplar 6 for the Light, Moderate, and Extensive categories of purposeful experience.

Multiple Comparisons

Dependent Variable: Proximity to 6

						95% Confidence Interval	
						Interval	
	(I) Purposeful Experience	(J) Purposeful Experience	Mean Difference (I-J)	Std. Error	Sig.	Lower Bound	Upper Bound
Dunnett T3	Light	Moderate	-1.91	1.012	.177	-4.38	.56
		Extensive	-.84	1.049	.810	-3.39	1.72
	Moderate	Light	1.91	1.012	.177	-.56	4.38
		Extensive	1.07	.892	.547	-1.10	3.25
	Extensive	Light	.84	1.049	.810	-1.72	3.39
		Moderate	-1.07	.892	.547	-3.25	1.10

Based on observed means.

The error term is Mean Square(Error) = 20.602.

Figure 83. Post hoc pair-wise comparisons among the categories of purposeful experience for proximity to Exemplar 6.

Note in Figure 84 that higher proximity values indicate that a participant's sort has low similarity with the exemplar; while lower proximity values indicate the participant sort is more similar to the exemplar. Examination of the mean proximities for each level of purposeful experience indicate that proximity to the exemplar increased, becoming less similar to the exemplar, for participants with more than light programming experience. However, students with extensive experience had lower proximity values than those with moderate experience. Sorts of students with the least programming

experience had the closest proximity to the exemplar while sorts of students with a moderate amount of programming experience had the farthest proximities to the exemplar.

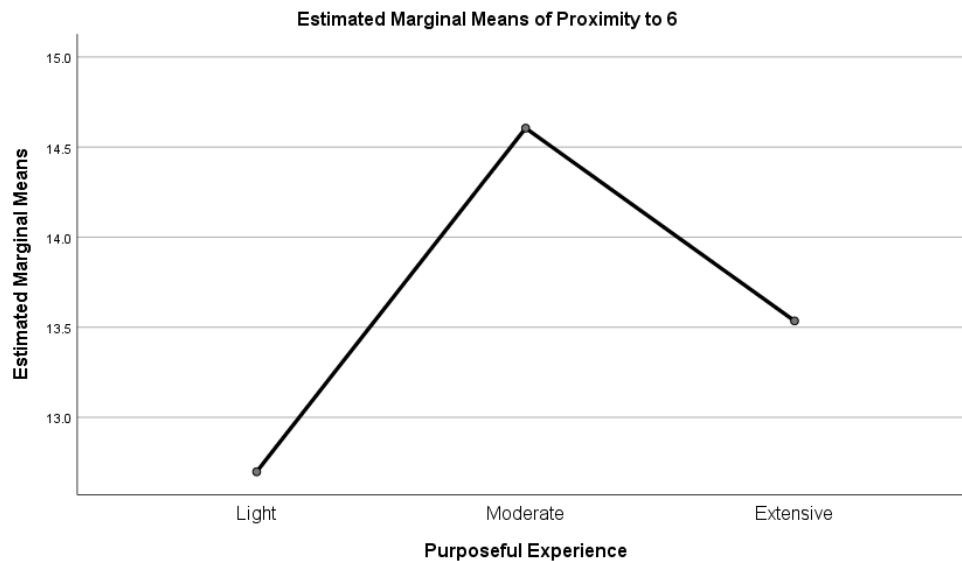


Figure 84. Line chart of mean proximity values to Exemplar 6 by categories of purposeful experience.

Results Statistical Significance for Proximity to Exemplar Sort 21

A one-way ANOVA compared participants' sort proximity to exemplar sort 21 grouped by level of participant programming experience (Figure 85). For the entire model, a statistically significant difference was found based on category of purposeful programming experience, $F(2,121) = 8.04$ $p = .001$, $\eta^2 = .12$. The effect size was $\eta^2 = .12$; a medium to large effect size (Kirk, 1996) indicating that 12% of the variance of proximities to exemplar sort 21 was explained by a student's level of programming experience.

Tests of Between-Subjects Effects

Dependent Variable: Proximity to 21

Source	Type III Sum of Squares	df	Mean Square	F	Sig.	Partial Eta Squared
Corrected Model	104.965 ^a	2	52.482	8.039	.001	.117
Intercept	22412.290	1	22412.290	3433.175	.000	.966
practiceCategory	104.965	2	52.482	8.039	.001	.117
Error	789.906	121	6.528			
Total	23386.000	124				
Corrected Total	894.871	123				

a. R Squared = .117 (Adjusted R Squared = .103)

Figure 85. One-way ANOVA comparing proximities to Exemplar sort 21 by category of purposeful experience.

The analysis revealed the following means and standard deviations (Figure 86) for each level of programming experience: a) Light ($m = 12.37$, $sd = 3.12$), b) Moderate ($m = 13.45$, $sd = 2.27$), and c) Extensive ($m = 14.58$, $sd = 2.13$). Post hoc tests using Bonferroni found a statistically significant difference in the means of the Light experience group to Extensive groups ($p < .001$) as shown in Figure 87.

Descriptive Statistics

Dependent Variable: Proximity to 21

Purposeful Experience	Mean	Std. Deviation	N
Light	12.37	3.117	43
Moderate	13.45	2.274	38
Extensive	14.58	2.130	43
Total	13.47	2.697	124

Figure 86. Mean and standard deviation statistics of proximities to Exemplar 21 for the Light, Moderate, and Extensive categories of purposeful experience.

Multiple Comparisons

Dependent Variable: Proximity to 21

					95% Confidence Interval		
(I) Purposeful Experience	(J) Purposeful Experience	Mean Difference (I-J)	Std. Error	Sig.	Lower Bound	Upper Bound	
Bonferroni Light	Moderate	-1.08	.569	.183	-2.46	.31	
	Extensive	-2.21*	.551	.000	-3.55	-.87	
	Moderate	Light	1.08	.569	.183	-.31	2.46
	Extensive	-1.13	.569	.145	-2.52	.25	
	Extensive	Light	2.21*	.551	.000	.87	3.55
	Moderate	1.13	.569	.145	-.25	2.52	

Based on observed means.

The error term is Mean Square(Error) = 6.528.

*. The mean difference is significant at the .05 level.

Figure 87. Post hoc pair-wise comparisons among the categories of purposeful experience for proximity to Exemplar 21.

Note in Figure 88 that higher proximity values indicate that a participant's sort has low similarity with the exemplar; while lower proximity values indicate the participant sort is more similar to the exemplar. Examination of the mean proximities for each level of purposeful experience indicate that proximity to the exemplar increased, becoming less similar to the exemplar, relative to progressive levels of programming experience (light, moderate, and extensive). Participants with light programming experience had sorts with the nearest proximity to the exemplar while students with extensive programming experience had sorts with the furthest proximities to the exemplar.

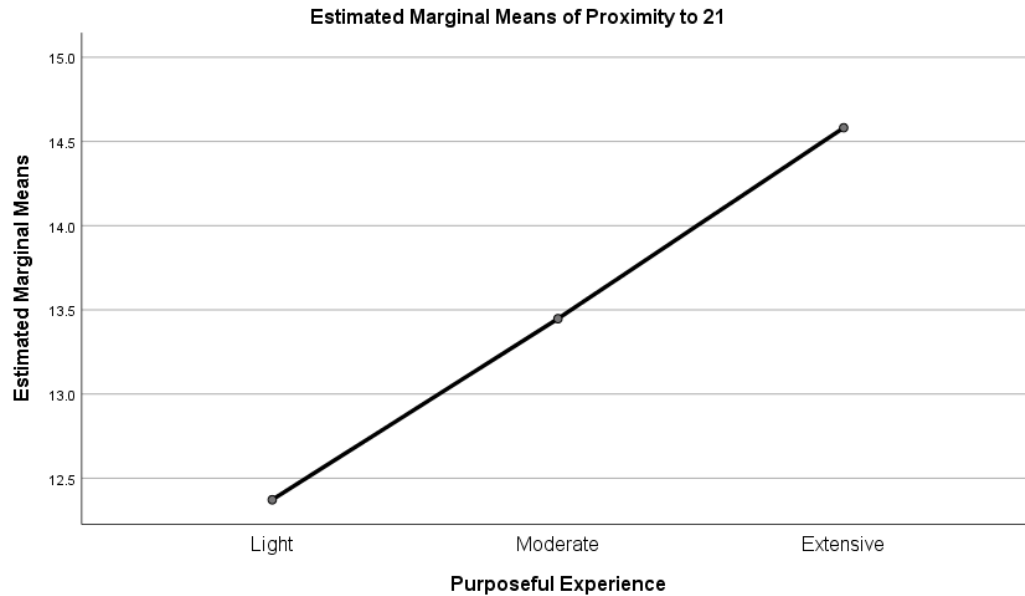


Figure 88. Line chart of mean proximity values to Exemplar 21 by categories of purposeful experience.

Results Statistical Significance for Proximity to Exemplar Sort 185

A one-way ANOVA compared participants' sort proximity to exemplar sort 185 grouped by level of participant programming experience (Figure 89). For the entire model, a statistically significant difference was found based on category of purposeful programming experience, $F(2,121) = 3.23$, $p = .043$, $\eta^2 = .05$. The effect size was $\eta^2 = .05$; a small effect size (Kirk, 1996) indicating that 5% of the variance of proximities to exemplar sort 185 was explained by a student's level of programming experience.

Tests of Between-Subjects Effects

Dependent Variable: Proximity to 185

Source	Type III Sum of Squares	df	Mean Square	F	Sig.	Partial Eta Squared
Corrected Model	29.578 ^a	2	14.789	3.228	.043	.051
Intercept	19925.532	1	19925.532	4348.973	.000	.973
practiceCategory	29.578	2	14.789	3.228	.043	.051
Error	554.381	121	4.582			
Total	20589.000	124				
Corrected Total	583.960	123				

a. R Squared = .051 (Adjusted R Squared = .035)

Figure 89. One-way ANOVA comparing proximities to Exemplar sort 185 by category of purposeful experience.

The analysis revealed the following means and standard deviations (Figure 90) for each level of programming experience: a) Light ($m = 12.16$, $sd = 2.13$), b) Moderate ($m = 12.61$, $sd = 2.31$), and c) Extensive ($m = 13.33$, $sd = 2.0$). Post hoc tests using Bonferroni found a statistically significant difference in the means of participants with Light programming experience compared to those with Extensive experience ($p = .039$) as shown in Figure 91.

Descriptive Statistics

Dependent Variable: Proximity to 185

Purposeful Experience	Mean	Std. Deviation	N
Light	12.16	2.126	43
Moderate	12.61	2.308	38
Extensive	13.33	1.997	43
Total	12.70	2.179	124

Figure 90. Mean and standard deviation statistics of proximities to Exemplar 185 for the Light, Moderate, and Extensive categories of purposeful experience.

Multiple Comparisons

Dependent Variable: Proximity to 185

					95% Confidence Interval		
(I) Purposeful Experience	(J) Purposeful Experience	Mean Difference (I-J)	Std. Error	Sig.	Lower Bound	Upper Bound	
Bonferroni Light	Moderate	-.44	.477	1.000	-1.60	.71	
	Extensive	-1.16*	.462	.039	-2.28	-.04	
	Moderate	Light	.44	.477	1.000	-.71	1.60
	Extensive	-.72	.477	.400	-1.88	.44	
	Extensive	Light	1.16*	.462	.039	.04	2.28
	Moderate	.72	.477	.400	-.44	1.88	

Based on observed means.

The error term is Mean Square(Error) = 4.582.

*. The mean difference is significant at the .05 level.

Figure 91. Post hoc pair-wise comparisons among the categories of purposeful experience for proximity to Exemplar 185.

Note in Figure 92 that higher proximity values indicate that a participant's sort has low similarity with the exemplar; while lower proximity values indicate the participant sort is more similar to the exemplar. Examination of the mean proximities for each level of purposeful experience indicate that proximity to the exemplar increased, becoming less similar to the exemplar, relative to progressive levels of programming experience (light, moderate, and extensive). Participants with the light programming experience had sorts with the nearest proximity to the exemplar while student with extensive programming experience had sorts with the furthest proximities to the exemplar.

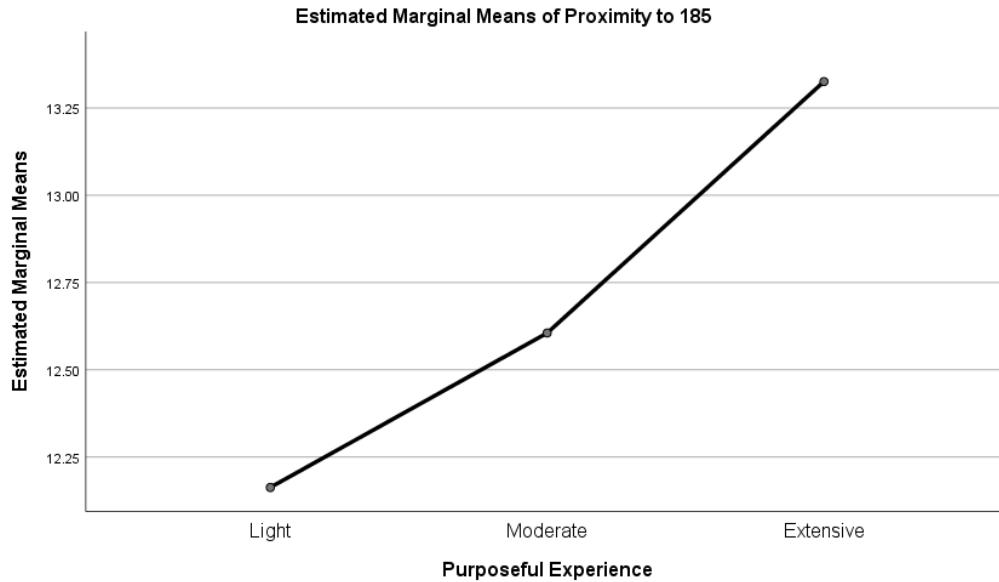


Figure 92. Line chart of mean proximity values to Exemplar 185 by categories of purposeful experience.

Results Statistical Significance for Proximity to Exemplar Sort 201

A one-way ANOVA compared participants' sort proximity to exemplar sort 201 grouped by level of participant programming experience (Figure 93). For the entire model, no statistically significant difference was found based on category of purposeful programming experience, $F(2,121) = 2.06, p > .05, \eta^2 = .03$. The effect size was $\eta^2 = .03$; a small effect size (Kirk, 1996) indicating that 3% of the variance of proximities to exemplar sort 201 was explained by a student's level of programming experience.

Tests of Between-Subjects Effects

Dependent Variable: Proximity to 201

Source	Type III Sum of Squares	df	Mean Square	F	Sig.	Partial Eta Squared
Corrected Model	76.410 ^a	2	38.205	2.062	.132	.033
Intercept	20901.381	1	20901.381	1127.845	.000	.903
practiceCategory	76.410	2	38.205	2.062	.132	.033
Error	2242.389	121	18.532			
Total	23145.000	124				
Corrected Total	2318.798	123				

a. R Squared = .033 (Adjusted R Squared = .017)

Figure 93. One-way ANOVA comparing proximities to Exemplar sort 201 by category of purposeful experience.

The analysis revealed the following means and standard deviations (Figure 94) for each level of programming experience: a) Light ($m = 12.33$, $sd = 4.93$), b) Moderate ($m = 14.13$, $sd = 3.77$), and c) Extensive ($m = 12.56$, $sd = 4.34$). Post hoc tests using Dunnett T3 found no statistically significant difference among the groups (Figure 95).

Descriptive Statistics

Dependent Variable: Proximity to 201

Purposeful Experience	Mean	Std. Deviation	N
Light	12.33	4.932	43
Moderate	14.13	3.772	38
Extensive	12.56	4.067	43
Total	12.96	4.342	124

Figure 94. Mean and standard deviation statistics of proximities to Exemplar 201 for the Light, Moderate, and Extensive categories of purposeful experience.

Multiple Comparisons

Dependent Variable: Proximity to 201

						95% Confidence Interval	
						Lower	Upper
	(I) Purposeful Experience	(J) Purposeful Experience	Mean Difference (I-J)	Std. Error	Sig.	Bound	Bound
Dunnett T3	Light	Moderate	-1.81	.970	.185	-4.17	.56
		Extensive	-.23	.975	.993	-2.61	2.14
	Moderate	Light	1.81	.970	.185	-.56	4.17
		Extensive	1.57	.871	.206	-.55	3.70
	Extensive	Light	.23	.975	.993	-2.14	2.61
		Moderate	-1.57	.871	.206	-3.70	.55

Based on observed means.

The error term is Mean Square(Error) = 18.532.

Figure 95. Post hoc pair-wise comparisons among the categories of purposeful experience for proximity to Exemplar 201.

Note in Figure 96 that higher proximity values indicate that a participant's sort has low similarity with the exemplar; while lower proximity values indicate the participant sort is more similar to the exemplar. Examination of the mean proximities for each level of purposeful experience indicate that proximity to the exemplar increased, becoming less similar to the exemplar, for participants with more than light programming experience. However, students with extensive experience had lower proximity values than those with moderate experience. Sorts of students with the least programming experience had the closest proximity to the exemplar while sorts of students with a moderate amount of programming experience had the farthest proximities to the exemplar.

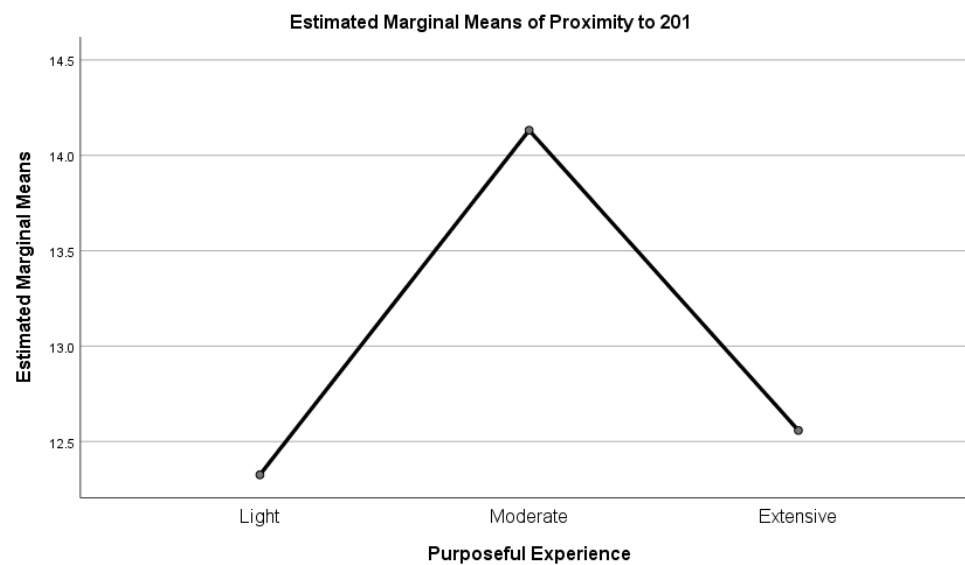


Figure 96. Line chart of mean proximity values to Exemplar 201 by categories of purposeful experience.

APPENDIX G

Statistical Analysis for Question 1 Redux

Research Question

Is there a statistically significant difference among non-introductory participants who submitted at least two card sorts categorized into thirds (Top, Average, Bottom) according to cumulative computer science grade point averages?

Null Hypothesis

H₀: There is no statistically significant difference among non-introductory participants who submitted at least two cards sorts categorized into thirds (Top, Average, Bottom) according to cumulative computer science grade point averages?

Assumption Testing

All assumptions were met. There were no missing data (Figure 97) and no outliers (Figure 98). Kolmogorov-Smirnov statistics indicated normally distributed data (Figure 99). Levene's statistic indicated homogeneity of variance (Figure 100) among the three categories of GPA Ranking for non-introductory participants who submitted at least two card sorts.

Case Processing Summary

		Valid		Cases Missing		Total	
GPA Ranking		N	Percent	N	Percent	N	Percent
nmst	Top	16	100.0%	0	0.0%	16	100.0%
	Average	20	100.0%	0	0.0%	20	100.0%
	Bottom	12	100.0%	0	0.0%	12	100.0%

Figure 97. Case Processing Summary for non-introductory participants categorized by GPA ranking. There are no missing cases.

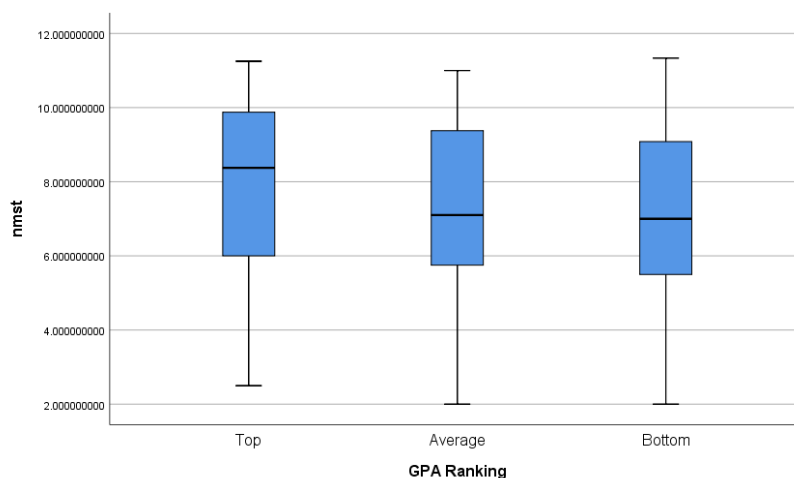


Figure 98. Box and whiskers plots for NMST means by GPA ranking categories. No outliers are observed.

Tests of Normality							
	GPA Ranking	Kolmogorov-Smirnov ^a			Shapiro-Wilk		
		Statistic	df	Sig.	Statistic	df	Sig.
nmst	Top	.142	16	.200*	.956	16	.592
	Average	.129	20	.200*	.947	20	.329
	Bottom	.188	12	.200*	.924	12	.319

*. This is a lower bound of the true significance.

a. Lilliefors Significance Correction

Figure 99. Test of Normality of NMST means by GPA ranking categories. Kolmogorov-Smirnov statistics are greater than .05 for each group indicating normal distributions.

Test of Homogeneity of Variance					
		Levene Statistic	df1	df2	Sig.
nmst	Based on Mean	.017	2	45	.983
	Based on Median	.004	2	45	.996
	Based on Median and with adjusted df	.004	2	42.752	.996
	Based on trimmed mean	.011	2	45	.989

Figure 100. Test of Homogeneity of Variance among the groups of GPA categories. Levene's statistic is greater than .05 indicating the assumption has been met.

Results of Statistical Significance

A one-way factorial analysis was conducted to compare non-introductory participants who submitted at least two card sorts categorized into thirds (Top, Average,

Bottom) according to cumulative computer science grade point averages. No statistically significant difference on the entire model was found between the categories of GPA Ranking, $[F(2, 45) = .20, p = .82, \eta^2 = .009]$ as shown in Figure 101. Although not statistically significant, the trend for mean values of NMST as shown in Figure 102 shows an increase with each step increase, from Bottom to Top, in GPA ranking category. This trend is visualized in Figure 103.

Tests of Between-Subjects Effects

Dependent Variable: nmst

Source	Type III Sum of Squares	df	Mean Square	F	Sig.	Partial Eta Squared
Corrected Model	2.570 ^a	2	1.285	.200	.820	.009
Intercept	2539.069	1	2539.069	394.464	.000	.898
gpa	2.570	2	1.285	.200	.820	.009
Error	289.654	45	6.437			
Total	2959.573	48				
Corrected Total	292.224	47				

a. R Squared = .009 (Adjusted R Squared = -.035)

Figure 101. One-way ANOVA comparing non-introductory participants who submitted at least two card sorts categorized into thirds (Top, Average, Bottom) according to cumulative computer science grade point averages. No statistically significant difference is indicated.

Descriptive Statistics

Dependent Variable: nmst

GPA Ranking	Mean	Std. Deviation	N
Top	7.74791666869	2.49868021712	16
Average	7.40500001910	2.43032801664	20
Bottom	7.14583329400	2.75976795960	12
Total	7.45451388769	2.49349683817	48

Figure 102. Mean and standard deviation statistics of NMST values for the Bottom, Average, and Top categories of GPA ranking.

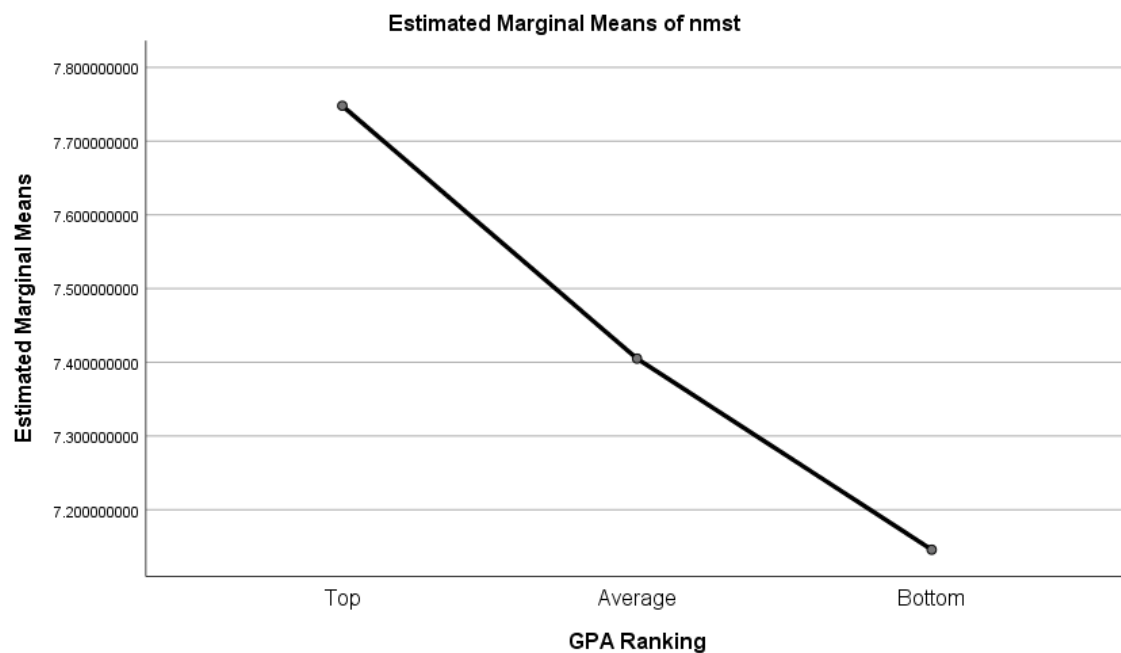


Figure 103. Line chart of mean NMST values for the Top, Average, and Bottom categories of GPA ranking.

APPENDIX H

Data Transformation Mappings

Table 9

Demographics

Survey Question Number	Survey Prompt	Survey Choice	Assigned Value	Interpretation
1	Classification level	Freshman	1	Freshman
		Sophomore	2	Sophomore
		Junior	3	Junior
		Senior	4	Senior
		Graduate	5	Graduate
2	Intended Major	CS	1	CS
		SE	2	SE
		DF	3	DF
		CS Minor	4	Non-CS
		Masters	5	Other
		Non-CS	4	Non-CS
		Open Response	5	Other
13	Year of Birth	Open Response	2019 - response	Age in years
14	Gender	Female	1	Female
		Male	2	Male
		No Response	0	Non-Binary

		Open response	-1	Non-Binary
15	Ethnicity	Asian	1	Asian
		Black	2	Black
		Hispanic	3	Hispanic
		Indian	5	Other
		Middle Eastern	5	Other
		White	4	White
		Open Response	5	Other

Table 10

Self-Assessments

Survey Question Number	Survey Prompt	Survey Choice	Assigned Value	Interpretation
3	Experience completing programming assignments			
		Complete in < 2 hours	5	< 2 hrs
		Complete in < 3 hours	4	< 3 hrs
		Struggle but complete > 3 hours	3	> 3 hrs
		Need help to complete	2	Need Help
		Don't get it, hate to code	1	Don't get it
4	Competence as a programmer			
		I can't code	1	Can't code
		Beginner	2	Beginner
		Confident I can complete assignments	3	Confident
		Enjoy programming; have few problems	4	Enjoy
		Really good; others seek my help	5	Really Good

Table 11

Prior Programming Instruction

Survey Question Number	Survey Prompt	Survey Choice	Assigned Value	Points Assigned
5	Experience programming micro-devices	Yes/No	1 or 0	1 or 0
6	Formal School Training (all that apply)			
6a	Elementary	Yes/No	1 or 0	1 or 0
6b	Middle	Yes/No	1 or 0	1 or 0
6c	HS Freshman	Yes/No	1 or 0	1 or 0
6d	HS Sophomore	Yes/No	1 or 0	1 or 0
6e	HS Junior	Yes/No	1 or 0	1 or 0
6f	HS Senior	Yes/No	1 or 0	1 or 0
6g	2 Year College	Yes/No	1 or 0	1 or 0
6h	Other College	Yes/No	1 or 0	1 or 0
7	Formal training for a job	Yes/No	1 or 0	1 or 0
8	Informal training			
8a	For a job	Yes/No	1 or 0	1 or 0
8b	For personal interest	Yes/No	1 or 0	1 or 0

Table 12

Purposeful Programming Experience

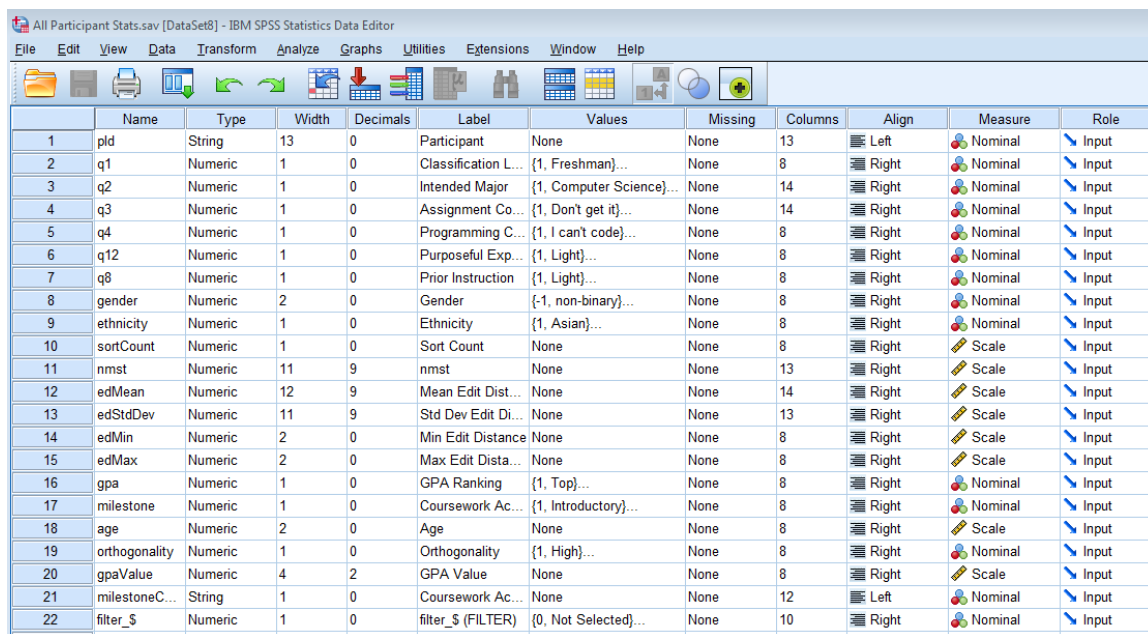
Survey Question Number	Survey Prompt	Survey Choice	Assigned Value	Points Assigned
9	Programming micro-devices	None	0	0
		Minimal	1	0.5
		Long time ago	2	1.0
		Ready	3	1.5
		Very Comfortable	4	2.0
10	Creating web pages	None	0	0
		Static pages	1	1
		Modify HTML	2	2
		Modify CSS	3	3
		Write Javascript	4	4
11	Creating web sites	None	0	0
		Simple sites	1	1.2
		Tool-based sites	2	2.4
		Interact with server	3	3.6
		Interact with cloud	4	4.8
12	Using code project services			

None	0	0
Retrieve apps	1	1.5
Search for code	2	3.0
Have own Library	3	4.5
Use with team	4	6.0

APPENDIX I

SPSS Variable Definitions for ParticipantStats dataset

Screenshot from SPSS:



	Name	Type	Width	Decimals	Label	Values	Missing	Columns	Align	Measure	Role
1	pld	String	13	0	Participant	None	None	13	Left	Nominal	Input
2	q1	Numeric	1	0	Classification L...	{1, Freshman}...	None	8	Right	Nominal	Input
3	q2	Numeric	1	0	Intended Major	{1, Computer Science}...	None	14	Right	Nominal	Input
4	q3	Numeric	1	0	Assignment Co...	{1, Don't get it}...	None	14	Right	Nominal	Input
5	q4	Numeric	1	0	Programming C...	{1, I can't code}...	None	8	Right	Nominal	Input
6	q12	Numeric	1	0	Purposeful Exp...	{1, Light}...	None	8	Right	Nominal	Input
7	q8	Numeric	1	0	Prior Instruction	{1, Light}...	None	8	Right	Nominal	Input
8	gender	Numeric	2	0	Gender	{-1, non-binary}...	None	8	Right	Nominal	Input
9	ethnicity	Numeric	1	0	Ethnicity	{1, Asian}...	None	8	Right	Nominal	Input
10	sortCount	Numeric	1	0	Sort Count	None	None	8	Right	Scale	Input
11	nmst	Numeric	11	9	nmst	None	None	13	Right	Scale	Input
12	edMean	Numeric	12	9	Mean Edit Dist...	None	None	14	Right	Scale	Input
13	edStdDev	Numeric	11	9	Std Dev Edit Di...	None	None	13	Right	Scale	Input
14	edMin	Numeric	2	0	Min Edit Distance	None	None	8	Right	Scale	Input
15	edMax	Numeric	2	0	Max Edit Dista...	None	None	8	Right	Scale	Input
16	gpa	Numeric	1	0	GPA Ranking	{1, Top}...	None	8	Right	Nominal	Input
17	milestone	Numeric	1	0	Coursework Ac...	{1, Introductory}...	None	8	Right	Nominal	Input
18	age	Numeric	2	0	Age	None	None	8	Right	Scale	Input
19	orthogonality	Numeric	1	0	Orthogonality	{1, High}...	None	8	Right	Nominal	Input
20	gpaValue	Numeric	4	2	GPA Value	None	None	8	Right	Scale	Input
21	milestoneC...	String	1	0	Coursework Ac...	None	None	12	Left	Nominal	Input
22	filter_\$	Numeric	1	0	filter_\$ (FILTER)	{0, Not Selected}...	None	10	Right	Nominal	Input

APPENDIX J

Descriptive Statistics SPSS Output for ParticipantStats

```

DATASET NAME DataSet1 WINDOW=FRONT.
FREQUENCIES VARIABLES=milestoneCode milestone q1 q2 nmst orthogonality gpaValue gpa gender
    ethnicity q12 q8 q3 q4 sortCount
    /NTILES=3
    /BARCHART FREQ
    /ORDER=ANALYSIS.

```

Frequencies

[DataSet1] Y:\MEM Documents\EdD\14- Dissertation\Chapter 4\All Participant Stats.sav

		Statistics			
		Coursework Achievement Level	Coursework Achievement Category	Classification Level	Intended Major
N	Valid	135	135	135	135
	Missing	0	0	0	0
Percentiles	33.33333333		1.00	2.00	1.00
	66.66666667		2.00	4.00	3.00

		Statistics				
		nmst	Orthogonality	GPA Value	GPA Ranking	Gender
N	Valid	124	135	85	135	135
	Missing	11	0	50	0	0
Percentiles	33.33333333	.0000000000	2.00	2.6833	2.00	2.00
	66.66666667	6.888888995	3.00	3.4000	2.00	2.00

		Statistics			
		Ethnicity	Purposeful Experience	Prior Instruction	Assignment Completion
N	Valid	135	135	135	135
	Missing	0	0	0	0
Percentiles	33.33333333	2.33	1.00	2.00	3.00
	66.66666667	6.00	3.00	3.00	4.00

		Statistics	
		Programming Competence	Sort Count
N	Valid	135	124
	Missing	0	11
Percentiles	33.33333333	2.00	1.00
	66.66666667	3.00	4.00

Frequency Table

Coursework Achievment Level

		Frequency	Percent	Valid Percent	Cumulative Percent
Valid	A	52	38.5	38.5	38.5
	B	14	10.4	10.4	48.9
	C	38	28.1	28.1	77.0
	D	4	3.0	3.0	80.0
	E	27	20.0	20.0	100.0
	Total	135	100.0	100.0	

Coursework Achievment Category

		Frequency	Percent	Valid Percent	Cumulative Percent
Valid	Introductory	52	38.5	38.5	38.5
	Mid Program	52	38.5	38.5	77.0
	Completing	31	23.0	23.0	100.0
	Total	135	100.0	100.0	

Classification Level

		Frequency	Percent	Valid Percent	Cumulative Percent
Valid	Freshman	20	14.8	14.8	14.8
	Sophomore	29	21.5	21.5	36.3
	Junior	30	22.2	22.2	58.5
	Senior	56	41.5	41.5	100.0
	Total	135	100.0	100.0	

Intended Major

		Frequency	Percent	Valid Percent	Cumulative Percent
Valid	Computer Science	62	45.9	45.9	45.9
	Software Engineering	22	16.3	16.3	62.2
	Digital Forensics	9	6.7	6.7	68.9
	CS Minor	3	2.2	2.2	71.1
	non-CS	35	25.9	25.9	97.0
	Other	4	3.0	3.0	100.0
	Total	135	100.0	100.0	

nmst

		Frequency	Percent	Valid Percent	Cumulative Percent
Valid	.000000000	56	41.5	45.2	45.2
	2.000000000	2	1.5	1.6	46.8
	2.500000000	2	1.5	1.6	48.4
	3.000000000	1	.7	.8	49.2
	4.250000000	1	.7	.8	50.0
	4.500000000	1	.7	.8	50.8
	5.000000000	4	3.0	3.2	54.0
	5.500000000	3	2.2	2.4	56.5
	5.599999905	1	.7	.8	57.3
	5.750000000	3	2.2	2.4	59.7
	6.000000000	2	1.5	1.6	61.3
	6.250000000	3	2.2	2.4	63.7
	6.500000000	2	1.5	1.6	65.3
	6.666666508	1	.7	.8	66.1
	6.833333492	1	.7	.8	66.9
	7.000000000	6	4.4	4.8	71.8
	7.199999809	1	.7	.8	72.6
	7.250000000	1	.7	.8	73.4
	7.666666508	1	.7	.8	74.2
	8.000000000	3	2.2	2.4	76.6
	8.250000000	2	1.5	1.6	78.2
	8.500000000	2	1.5	1.6	79.8
	8.750000000	1	.7	.8	80.6
	8.800000191	2	1.5	1.6	82.3

nmst

		Frequency	Percent	Valid Percent	Cumulative Percent
	9.000000000	1	.7	.8	83.1
	9.250000000	1	.7	.8	83.9
	9.500000000	3	2.2	2.4	86.3
	9.666666985	2	1.5	1.6	87.9
	9.750000000	1	.7	.8	88.7
	10.000000000	2	1.5	1.6	90.3
	10.250000000	2	1.5	1.6	91.9
	10.500000000	4	3.0	3.2	95.2
	10.750000000	1	.7	.8	96.0
	11.000000000	2	1.5	1.6	97.6
	11.250000000	2	1.5	1.6	99.2
	11.333333020	1	.7	.8	100.0
	Total	124	91.9	100.0	
Missing	System	11	8.1		
Total		135	100.0		

Orthogonality

		Frequency	Percent	Valid Percent	Cumulative Percent
Valid	Top	41	30.4	30.4	30.4
	Average	27	20.0	20.0	50.4
	Bottom	67	49.6	49.6	100.0
	Total	135	100.0	100.0	

GPA Ranking

		Frequency	Percent	Valid Percent	Cumulative Percent
Valid	Top	29	21.5	21.5	21.5
	Average	78	57.8	57.8	79.3
	Bottom	28	20.7	20.7	100.0
	Total	135	100.0	100.0	

Gender

		Frequency	Percent	Valid Percent	Cumulative Percent
Valid	non-binary	2	1.5	1.5	1.5
	Declined	1	.7	.7	2.2
	Female	32	23.7	23.7	25.9
	Male	100	74.1	74.1	100.0
	Total	135	100.0	100.0	

Ethnicity

		Frequency	Percent	Valid Percent	Cumulative Percent
Valid	Asian	22	16.3	16.3	16.3
	Black	23	17.0	17.0	33.3
	Hispanic	27	20.0	20.0	53.3
	Indian	1	.7	.7	54.1
	Middle Eastern	1	.7	.7	54.8
	White	56	41.5	41.5	96.3
	Other	5	3.7	3.7	100.0
	Total	135	100.0	100.0	

Purposeful Experience

		Frequency	Percent	Valid Percent	Cumulative Percent
Valid	Light	48	35.6	35.6	35.6
	Moderate	41	30.4	30.4	65.9
	Extensive	46	34.1	34.1	100.0
	Total	135	100.0	100.0	

Prior Instruction

		Frequency	Percent	Valid Percent	Cumulative Percent
Valid	Light	40	29.6	29.6	29.6
	Moderate	49	36.3	36.3	65.9
	Extensive	46	34.1	34.1	100.0
	Total	135	100.0	100.0	

Assignment Completion

		Frequency	Percent	Valid Percent	Cumulative Percent
Valid	Don't get it	6	4.4	4.4	4.4
	Need Help to complete	19	14.1	14.1	18.5
	Struggle > 3 hrs	56	41.5	41.5	60.0
	Complete in < 3 hrs	46	34.1	34.1	94.1
	Complete in < 2 hrs	8	5.9	5.9	100.0
	Total	135	100.0	100.0	

Programming Competence

		Frequency	Percent	Valid Percent	Cumulative Percent
Valid	I can't code	9	6.7	6.7	6.7
	Beginner	47	34.8	34.8	41.5
	Confident	50	37.0	37.0	78.5
	Enjoy- few problems	22	16.3	16.3	94.8
	Really good	7	5.2	5.2	100.0
	Total	135	100.0	100.0	

Sort Count

		Frequency	Percent	Valid Percent	Cumulative Percent
Valid	1	56	41.5	45.2	45.2
	2	17	12.6	13.7	58.9
	3	9	6.7	7.3	66.1
	4	35	25.9	28.2	94.4
	5	5	3.7	4.0	98.4
	6	1	.7	.8	99.2
	8	1	.7	.8	100.0
	Total	124	91.9	100.0	
Missing	System	11	8.1		
Total		135	100.0		

APPENDIX K

Program Listing for Edit Distance Calculation

```

editDistance.txt

'use strict';
// Version 1.0 of edit distance calculator for Study Portal using Async/Await -
2/17/19

const db = require('./db.js');
var munkres = require('munkres-js');

//const munkres = require('munkres-js');
const sqlTxt = 'SELECT MAX("sortId") FROM "sortCaptures";';
    sqlTxt1 = 'SELECT * FROM "get_sort_groups" WHERE "sortId" = $1;';
    sqlTxt2 = 'SELECT * FROM "get_sort_groups" WHERE "sortId" = $1;';
    sqlTxt3 = 'SELECT add_edit_distance ($1, $2, $3, $4, $5);';

//      variables go here
var sortCount = 0;
    sortsTotal = 0,
    sortsSkipped = 0,
    edPairs = 0,
    startTime = Date.now();

/***** Main routine *****/
//var sqlTxt = 'SELECT MAX("sortId") FROM "sortCaptures";';

db.query(sqlTxt)
    .then(res => processOuterSorts(res.rows))
    .catch(e => setImmediate( () => {throw e }));

/***** Functions go here *****/

async function processOuterSorts(tRows) {
    sortsTotal = tRows[0].max; //      max sort ID
    console.log(Date().toString() + ' EditDistance V.1.0 - ' + sortsTotal
+ ' total sorts to process');
    var sqlValues = [];
    var oGroups = [];
    var oGrpCount;
    var outerId;

    for (var i = 1; i < sortsTotal; i++) { //      Stop prior to
last row, will be the inner sort
        sqlValues[0] = i.toString();

        var oResults = await db.query(sqlTxt1, sqlValues);
        var oRows = oResults.rows;
        oGrpCount = oRows.length;

        if (oGrpCount === 0) {
            console.log('Skipping Sort ' + (i) + ' - No groups');
            sortsSkipped++;
        } else {
            sortCount++;
            outerId = oRows[0].sortId;
            console.log('OuterSort = ' + outerId);
            console.log('Sort ' + outerId + ' has ' + oGrpCount + '
groups');

            oGroups = buildGroups(oRows);
            console.log('groups = ' + JSON.stringify(oGroups));

            //      Process the Inner Sorts
            for ( var j = outerId + 1; j <= sortsTotal; j++) {
                sqlValues = [];
                sqlValues[0] = j.toString();

                var iResults = await db.query(sqlTxt2,
sqlValues);

```

```

                                processInnerSort(iResults.rows, outerId,
oGrpCount, oGroups);
                                }
                                }
                                sortCount++;
inner sort // increment to account for the last
// Processing is complete
var endTime = Date.now(),
    runTime = (endTime - startTime)/1000;

runTime console.log(Date().toString() + ' EditDistance completed - Run Time = ' +
+ ' seconds');
console.log('Sorts skipped = ' + sortsSkipped);
console.log('Sorts processed = ' + sortCount);
console.log(edPairs + ' editDistance Pairs created');
}

function processInnerSort(iRows, oId, oGrpCnt, oGrps) {
    var iGrpCnt = iRows.length;
    var iGrps = [];

    var edge_weights = [[],[ ]],
        cost_weights = [],
        solution = [],
        editDistance;

    var weights_JSON,
        solution_JSON;

    var sqlValues = [];
    var promise;

    if (iGrpCnt === 0) {
// console.log('Sort ' + sortNum + ' has no groups');
        return;
    }
    var innerId = iRows[0].sortId;
// console.log(' innerSort = ' + innerId);

    iGrps = buildGroups(iRows);
// console.log('oGrps has ' + oGrps.length + ' rows - iGrps has ' +
iGrps.length + ' rows');

    if (oGrpCnt > iGrpCnt) {
iGrps); edge_weights = calculate_edge_weights(oGrpCnt, oGrps,
    } else {
oGrps); edge_weights = calculate_edge_weights(iGrpCnt, iGrps,
    };

    cost_weights = complement_edge_weights(edge_weights);
    weights_JSON = JSON.stringify(edge_weights);
// console.log('edge_weights = ' + weights_JSON);

    solution = munkres(cost_weights);
    solution_JSON = JSON.stringify(solution);

    editDistance = calculate_editDistance (solution, edge_weights);
// console.log('Edit Distance between ' + oId + '(Sort A) and ' +
innerId + '(Sort B) = ' + editDistance);
// console.log(' Solution is ' + solution_JSON);

    sqlValues[0] = oId.toString();
    sqlValues[1] = innerId.toString();

```

```

        sqlValues[2] = parseInt(editDistance);
        sqlValues[3] = solution_JSON;
        sqlValues[4] = weights_JSON;

        edPairs++;
        promise = db.query (sqlTxt3, sqlValues)           //      Insert
into the editDistance table
    }

function buildGroups(rows) {
    var xgroup = [];
    // console.log('Sort ' + rows[0].sortId + ' items = ' + rows[0].item);
    for (var i = 0; i < rows.length; i++) {
        xgroup[i] = rows[i].item;
        for (var j = 0; j < 26; j++) {
            // for (var j = 0; j < rows[i].item.length; j++) {
                if (xgroup[i][j] == (i + 1)) {
                    xgroup[i][j] = '1';
                } else {
                    xgroup[i][j] = '0';
                }
            }
        }
    }
    return xgroup;
}

function calculate_edge_weights(grpCount, sortA, sortB) {
    var edge_weight_tbl = [],
        outer = [],
        inner = [];

    // Initialize the edge_weights to 0 elements in common
    for (var i = 0; i < grpCount; i++) {
        inner = [];
        for (var j = 0; j < grpCount; j++) {
            // edge_weight_tbl[[i],[j]] = 0;
            inner.push(0);
        }
        outer.push(inner);
    }
    edge_weight_tbl = outer;

    // Compare each group in sort A to each group in sort B for elements in
    common
    var weight = 0;
    for (var i = 0; i < sortA.length; i++) {
        for (var j = 0; j < sortB.length; j++) {
            weight = 0;
            for (var k = 0; k < 26; k++) {
                if (sortA[i][k] == '1' && sortB[j][k] == '1') {
                    weight++;
                }
            }
            edge_weight_tbl[i][j] = weight;
        }
    }
    return edge_weight_tbl;
}

function complement_edge_weights(weight_tbl) {
    var outer = [],
        inner = [];

    var dimension = weight_tbl[0].length;
    for (var i = 0; i < dimension; i++) {
        inner = [];
        for (var j = 0; j < dimension; j++) {

```



```

//          weight_tbl[i][j] = 26 - weight_tbl[i][j];
//          inner.push(26 - weight_tbl[i][j]);
//      };
//      outer.push(inner);
//  };
//  return outer;
// }

function calculate_editDistance (solution_vector, weights_tbl) {
    var metric = 26;
    var weight = 0;
    var row,
        col;

    for (var i = 0; i < solution_vector.length; i++) {
        row = solution_vector[i][0];
        col = solution_vector[i][1];
        weight = weights_tbl[row][col];
        metric = metric - weight;
    };
    return (metric);
}

```

Program Execution:

```

Your environment has been set up for using Node.js 8.11.4 (ia32) and npm.
Y:\MEM Documents\NodePgms>node editDistance/editDistance.js
Fri May 10 2019 10:47:08 GMT-0500 (Central Daylight Time) EditDistance U.1.0 -
302 total sorts to process
Skipping Sort 1 - No groups
Skipping Sort 2 - No groups
Skipping Sort 138 - No groups
Skipping Sort 139 - No groups
Skipping Sort 140 - No groups
Skipping Sort 141 - No groups
Fri May 10 2019 10:48:56 GMT-0500 (Central Daylight Time) EditDistance completed
- Run Time = 108.256 seconds
Sorts skipped = 6
Sorts processed = 296
43660 editDistance Pairs created

```

APPENDIX L

Program Listing for Participant Orthogonality Calculation

```

statsApp.js

'use strict';
// Version 1 of participant sort stats (orthogonality) calculator for Study
Portal using Async/Await - 2/25/19
/*
    Given a set of participants (1 or more) P
    Select the set S of sortIds for P
    Let k = S.length //number of vertices
    Select the set E of editDistances rows where sortA and sortB are members
of S
    Initialize the weightedGraph g for k vertices
    For I = 1 to S.length // for each editDistance pair in the set
        Add the edge to g from sortA to sortB with edweight
    Create anEagerPrim graph and its mst
    Initialize the sum of the weights of the minimum spanning tree, s
    For I = 1 to mst.length // traverse the minimum spanning tree
        Increment s by the weight of mst[i]
    Set NMST = s/k
*/

const db = require('./db.js');
const jStat = require('jStat').jStat;
var jsgraphs = require('js-graph-algorithms');

const sqlGet = 'SELECT DISTINCT ("Pid") FROM "sortCaptures";',
    sqlTxt = 'SELECT "sortId" FROM "sortCaptures" WHERE "Pid" = $1;',
    sqlGet1 = 'SELECT count (*) FROM "sortCaptures", "sortCategories" WHERE
"sortCaptures"."Pid" = $1',
    sqlGet2 = 'AND "sortCaptures"."sortId" = "sortCategories"."sortId" AND
"sortCategories"."catId" > 0;',
    sqlGetED = 'SELECT * FROM "get_participantSortEdges" ($1);',
    sqlAdd = 'SELECT "add_participantStats" ($1, $2, $3, $4, $5, $6, $7,
$8, $9, $10, $11, $12);';

var sqlValues = [];
var sqlGetGrps = sqlGet1 + sqlGet2;

//    variables go here

/***** Main routine *****/
//    Process each participant
console.log(Date().toString() + ' NMST v.1.03 - ');

//    Get the list of participants
db.query(sqlGet)
.then(res => processParticipants(res.rows))
.catch(e => setImmediate(() => {throw e}));

/***** Functions go here *****/
async function processParticipants(Prows) {
    var S = [],
        participant,
        participant ID
        nmst,
        //    NMST measure of orthogonality
        jEdges,
        //    JSON copy of edit distance pairs
        jMst;
        //    JSON copy of min spanning tree path
    var Pcount = Prows.length;
    console.log(Pcount + ' participants to process');

    //    Process each participant
    for (var j = 0; j < Pcount; j++) {
        var startTime = Date.now();
        participant = Prows[j].Pid;
        sqlValues = [];
    }
}

```

```

        sqlValues[0] = participant;
//      Get the number of sorts in the collection of sorts for this
participant
    S = await db.query(sqlTxt, sqlValues);
    var sRows = S.rows;
    var k = sRows.length;
//      Count of sorts in the collection (vertices)
    var g = new jsgraphs.WeightedGraph(k);
//      new graph dimensioned for vertex count
    console.log(k + ' total sorts to process for pId ' +
participant);
//      Get the total number of sort categories created by this
participant
    var grpResults = await db.query(sqlGetGrps, sqlValues); //
Get count of categories by this participant
    var grpRows = grpResults.rows;
    var sortGrpCount = grpRows[0].count;
//      console.log('participant ' + participant + ': sortGrpCount = ' +
sortGrpCount);

//      Extract the vertices of the collection from participant
sorts
    var vertices = [];
    for (var i = 0; i < k; i++) {
        vertices.push(sRows[i].sortId.toString());
//      console.log('Vertex ' + i + ' = ' + vertices[i]);
    };

//      Select the editDistance pairs (the edges) associated with
the sorts
    var qResult = await db.query(sqlGetED, sqlValues);
//      var qResult = await db.query(sqlTxt1, sqlValues);
    var edges = qResult.rows;
    jEdges = JSON.stringify(edges);
    var edgeCount = edges.length;
//      console.log(edgeCount + ' edges in the collection; jEdges = ' +
jEdges);

//      Add the edge weights for each vertices pair to the graph
//      var weights = [];
//      statistics vector of edit distances
    for (var i = 0; i < edgeCount; i++) {
g.addEdge(new jsgraphs.Edge(vertices.indexOf(edges[i].srtA), vertices.indexOf(edg
es[i].srtB), edges[i].edgeWeight));
        weights.push(edges[i].edgeWeight);
//      console.log('weights vector: ' + weights);
//      ***** Invoke the Prim algorithm to determine the
minimum spanning tree path *****
    var prim = new jsgraphs.EagerPrimMST(g);
    var mst = prim.mst;
    jMst = JSON.stringify(mst);

//      console.log('minimum spanning tree path:');
    var s = 0;
    for(var i=0; i < mst.length; ++i) {
        var e = mst[i];
        var v = e.either();
        var w = e.other(v);
//      console.log('(' + v + ', ' + w + '): ' + e.weight);
        s = s + e.weight;
    };
    nmst = s/k;

```

```

// ***** calculate descriptive statistics on the
edit distances *****
var stats = jStat(weights),
    edMin = 0,
    edMax = 0,
    edSum = 0,
    edMean = 0.0,
    edStdDev = 0.0,
    edCount = weights.length;
if (edCount > 0) {
    edMin = stats.min(),
    edMax = stats.max(),
    edSum = stats.sum(),
    edMean = stats.mean(),
    edStdDev = stats.stdev();
};

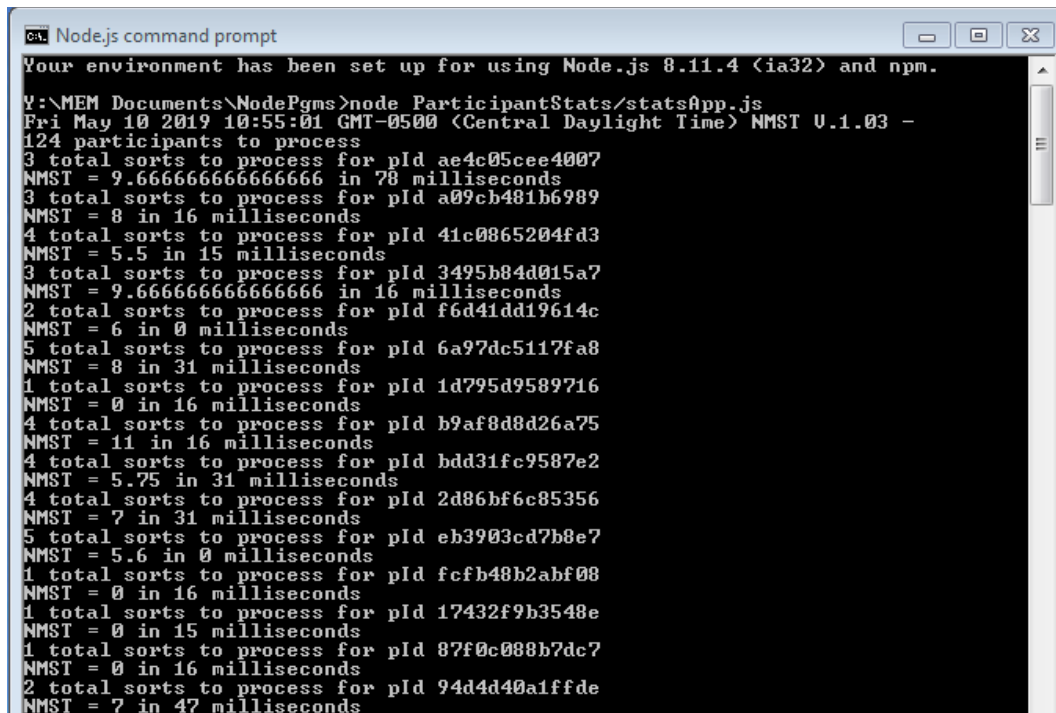
// console.log('edStats: cnt- ' + edCount + ' min- ' + edMin + '
max- ' + edMax + ' sum- ' + edSum + ' mean- ' + edMean + ' stDev- ' + edStdDev);

sqlValues[1] = k;
sqlValues[2] = sortGrpCount;
sqlValues[3] = nmst;
sqlValues[4] = jEdges;
sqlValues[5] = jMst;
sqlValues[6] = edSum;
sqlValues[7] = edMean;
sqlValues[8] = edStdDev;
sqlValues[9] = edCount;
sqlValues[10] = edMin;
sqlValues[11] = edMax;

var promise = await db.query(sqlAdd, sqlValues);
// Log the results in the Participant table
var runTime = (Date.now() - startTime);
console.log('NMST = ' + nmst + ' in ' + runTime + '
milliseconds');
}

```

Program Execution:



```

Node.js command prompt
Your environment has been set up for using Node.js 8.11.4 <ia32> and npm.

Y:\MEM Documents\NodePgms>node ParticipantStats/statsApp.js
Fri May 10 2019 10:55:01 GMT-0500 (Central Daylight Time) NMST U.1.03 -
124 participants to process
3 total sorts to process for pId ae4c05cee4007
NMST = 9.666666666666666 in 78 milliseconds
3 total sorts to process for pId a09cb481b6989
NMST = 8 in 16 milliseconds
4 total sorts to process for pId 41c0865204fd3
NMST = 5.5 in 15 milliseconds
3 total sorts to process for pId 3495b84d015a7
NMST = 9.666666666666666 in 16 milliseconds
2 total sorts to process for pId f6d41dd19614c
NMST = 6 in 0 milliseconds
5 total sorts to process for pId 6a97dc5117fa8
NMST = 8 in 31 milliseconds
1 total sorts to process for pId 1d795d9589716
NMST = 0 in 16 milliseconds
4 total sorts to process for pId b9af8d8d26a75
NMST = 11 in 16 milliseconds
4 total sorts to process for pId bdd31fc9587e2
NMST = 5.75 in 31 milliseconds
4 total sorts to process for pId 2d86bf6c85356
NMST = 7 in 31 milliseconds
5 total sorts to process for pId eb3903cd7b8e7
NMST = 5.6 in 0 milliseconds
1 total sorts to process for pId fcfb48b2abf08
NMST = 0 in 16 milliseconds
1 total sorts to process for pId 17432f9b3548e
NMST = 0 in 15 milliseconds
1 total sorts to process for pId 87f0c088b7dc7
NMST = 0 in 16 milliseconds
2 total sorts to process for pId 94d4d40a1ffde
NMST = 7 in 47 milliseconds

```

APPENDIX M

Structural Exemplar Calculations

Procedure: for each sort in the clique, sum its edit distances to all other sorts in the clique. The structural exemplar sort is the sort with the shortest total of edit distances (Deibel, Anderson, & Anderson, 2005).

Prominent sorts identified in collection 13 that are contained in cliques with more than two sorts (Table 13, Table 14, and Table 15):

Table 13

Clique sequence 109: (6, 58, 130, 194, 201, 206)

Sort:	6	58	130	194	201	206	Total
6	0	5	5	5	7	5	27
58	5	0	8	4	8	6	31
130	5	8	0	8	8	8	37
194	5	4	8	0	8	6	31
201	7	8	8	8	0	6	37
206	5	6	8	6	6	0	31

Note. d-size = 8, 6 sorts, nmst = 4.17

The Structural exemplar is sort 6.

Table 14

Clique sequence 68: (12, 13, 14, 15, 51)

Sort:	12	13	14	15	51	Total
12	0	2	6	4	6	18
13	2	0	5	4	7	18
14	6	5	0	2	8	21
15	4	4	2	0	8	18
51	6	7	8	8	0	29

Note. d-size = 8, 5 sorts, nmst = 2.8

There is a tie between sorts 12, 13, and 15. Based upon examination of the contents of these sorts, the Structural exemplar is 15.

Table 15

Clique sequence 110: (4, 21, 51, 74)

Sort:	4	15	21	51	74	Total
4	0	9	6	8	5	28
15	9	0	8	8	10	35
21	6	8	0	7	5	26
51	8	8	7	0	8	31
74	5	10	5	8	0	28

Note. d -size = 8, 4 sorts, nmst = 4.25 +

sequence 113: (15, 51, 21) d -size = 8, 3 sorts, nmst = 5

The Structural exemplar is sort 21.

APPENDIX N

Collections

Participants completed an online questionnaire that collected data such as their university classification level, intended major, experiences in purposeful programming outside of class, instruction in programming outside of the University, and self-assessments of their ability to complete programming assignments and their overall competency as a programmer. Appendix H lists the questions and how responses were coded in the database. An interactive tool was written to allow the researcher to select participants according to these data. Each selection of participants defined a specific cross-section of the sample. These were referred to as collections. Each collection was stored in the database and identified with a unique collection number. Queries were written to extract the statistics and card sorts for participants belonging to a specific collection number.

Fourteen collections in all were created. Some of these were created to explore the dataset and gain a general understanding of the data. Later on, collections were created in the search for a cross-section that most-likely represented participants who had attained the desired level of conceptual development. These collections are described in Chapter 4, in the section on RQ2 – Identification of card sorts that exemplify the desired level of conceptual development.

Following is the list of all fourteen collections, their selection criteria, and the resulting number of participants and sorts.

- Collection 1: All participants categorized as Completing. 79 sorts from 29 participants.

- Collection 2: All participants categorized as Introductory. 100 sorts from 50 participants.
- Collection 3: Introductory participants with a university classification of Freshman or Sophomore. 55 sorts from 32 participants. Comparison with Collection 2 indicates that 18 juniors or seniors were enrolled in the Introductory course.
- Collection 4: Juniors and Seniors categorized as Introductory with majors of Computer Science, Software Engineering Technology, or Digital Forensics. 17 sorts from 5 participants.
- Collection 5: Juniors and Seniors categorized as Introductory with intended majors categorized as Non-CS or Other. 28 sorts from 13 participants.
- Collection 6: Juniors and Seniors who self-reported as normally completing programming assignments in three hours or less, with a GPA in the Average or Top categories. 77 sorts from 27 participants.
- Collection 7: Participants with an intended major of Software Engineering Technology. 37 sorts from 18 participants.
- Collection 8: Seniors in the Top category of GPA ranking with majors of Computer Science, Software Engineering Technology, or Digital Forensics. 27 sorts from 13 participants.
- Collection 9: Juniors and Seniors with an NMST ranking in the Top or Average categories of GPA ranking with majors of Computer Science, Software Engineering Technology, or Digital Forensics. 107 sorts from 34 participants.

- Collection 10: All Juniors and Seniors who self-report their programming competency as Really Good or Enjoy with few problems. 48 sorts from 18 participants.
- Collection 11: Seniors with an NMST ranking in the Top category. 71 sorts from 20 participants.
- Collection 12: All Juniors and Seniors. 208 sorts from 79 participants.
- Collection 13: Juniors and Seniors who self-reported as completing programming assignments in 3 hours or less, and who are also either Confident in their programming competency, Enjoy programming with few problems, or are Really Good. 76 sorts from 29 participants.
- Collection 14: Juniors and Seniors who are not in Collection 13. 122 sorts from 46 participants.

APPENDIX O

IRB Approvals



Date: Nov 20, 2018 5:12 PM CST

TO: Michael Morrow

Li-Jen Lester

FROM: SHSU IRB

PROJECT TITLE: Using knowledge elicitation techniques to establish a baseline of quantitative measures of computational thinking skill acquisition among university computer science students

PROTOCOL #: IRB-2018-69

SUBMISSION TYPE: Initial

ACTION: Approved

DECISION DATE: November 20, 2018

EXPIRATION DATE: November 20, 2019

EXPEDITED REVIEW CATEGORY: 5. Research involving materials (data, documents, records, or specimens) that have been collected, or will be collected solely for nonresearch purposes (such as medical treatment or diagnosis).

7. Research on individual or group characteristics or behavior (including, but not limited to, research on perception, cognition, motivation, identity, language, communication, cultural beliefs or practices, and social behavior) or research employing survey, interview, oral history, focus group, program evaluation, human factors evaluation, or quality assurance methodologies.

Greetings,

The above-referenced submission has been reviewed by the IRB and it has been Approved. This decision expires on November 20, 2019. This approval is based on an appropriate risk/benefit ratio and a project design wherein the risks have been minimized. All research must be conducted in accordance with this approved submission.

Since Cayuse IRB does not currently possess the ability to provide a "stamp of approval" on any recruitment or consent documentation, it is the strong recommendation of this office to please include the following approval language in the footer of those recruitment and consent documents: IRB-2018-69/November 20, 2018/November 20, 2019.

Please remember that informed consent is a process beginning with a description of the project and insurance of participant understanding followed by a signed consent form. Informed consent must continue throughout the project via a dialogue between the researcher and research participant. Federal regulations require each participant receive a copy of the signed consent document.

Modifications: Please note that any revision to previously approved materials must be approved by this committee prior to initiation. Please submit a Modification Submission through [Cayuse IRB](#) for this procedure.

Incidents: All UNANTICIPATED PROBLEMS involving risks to subjects or others and SERIOUS and UNEXPECTED adverse events must be reported promptly to this office. Please submit an Incident Submission through [Cayuse IRB](#) for this procedure. All Department of Health and Human Services and sponsor reporting requirements should also be followed.

Renewals: Based on the risks, this project requires renewal reviews by this committee on an annual basis. Please submit a Renewal Submission through [Cayuse IRB](#) for this procedure. Your documentation for renewal must be received with sufficient time for review and updated approval before the expiration date of November 20, 2019.

Closures: When you have completed the project, a Closure Submission must be submitted through [Cayuse IRB](#) in order to close the project file.

Please note that all research records should be retained for a minimum of three years after the completion of the project.

If you have any questions, please contact the Sharla Miles at 936-294-4875 or irb@shsu.edu. Please include your protocol number in all correspondence with this committee.

Sincerely,

Donna Desforjes
IRB Chair, PHSC
PHSC-IRB



Date: March 25, 2019

TO: Michael Morrow

Li-Jen Lester

FROM: SHSU IRB

PROJECT TITLE: Using knowledge elicitation techniques to establish a baseline of quantitative measures of computational thinking skill acquisition among university computer science students

PROTOCOL #: IRB-2018-69

SUBMISSION TYPE: Modification

ACTION: Approved

DECISION DATE: March 20, 2019

EXPEDITED REVIEW CATEGORY: 5. Research involving materials (data, documents, records, or specimens) that have been collected, or will be collected solely for nonresearch purposes (such as medical treatment or diagnosis).

Greetings,

The above-referenced submission has been reviewed by the IRB and it has been Approved. This approval is based on an appropriate risk/benefit ratio and a project design wherein the risks have been minimized. All research must be conducted in accordance with this approved submission.

Please remember that informed consent is a process beginning with a description of the project and insurance of participant understanding followed by a signed consent form. Informed consent must continue throughout the project via a dialogue between the researcher and research participant. Federal regulations require each participant receive a copy of the signed consent document.

Modifications: Please note that any revision to previously approved materials must be approved by this committee prior to initiation. **Please submit a Modification Submission through [Cayuse IRB](#) for this procedure.**

Incidents: All UNANTICIPATED PROBLEMS involving risks to subjects or others and SERIOUS and UNEXPECTED adverse events must be reported promptly to this office. **Please submit an Incident Submission through [Cayuse IRB](#) for this procedure.** All Department of Health and Human Services and sponsor reporting requirements should also be followed.

Closures: When you have completed the project, a Closure Submission must be submitted through [Cayuse IRB](#) in order to close the project file.

Please note that all research records should be retained for a minimum of three years after the completion of the project.

If you have any questions, please contact the Sharla Miles at 936-294-4875 or irb@shsu.edu. Please include your protocol number in all correspondence with this committee.

Sincerely,

Donna M. Desforbes, Ph.D.

Chair, Committee for the Protection of Human Subjects

PHSC-IRB

VITA

M. Earnest Morrow

EDUCATION

Doctor of Education in Instructional Systems Design and Technology at Sam Houston State University, August 2015 – Present.

Master of Education in Instructional Technology at Sam Houston State University, August 2010 – May 2012.

Bachelor of Science in Business Administration (May 1972), Oklahoma State University, Stillwater, Oklahoma.

PROFESSIONAL EMPLOYMENT

Information Technology Professional, Exxon / ExxonMobil, Houston, TX, June 1976 – April 2009. Responsibilities included: systems analysis and design, systems support, leadership of projects, teams, and groups, systems administration, network administration, database administration, preparation and monitoring of budgets, evaluation of employee performance, recruiting and interviewing new college hires, advising management on information technology trends and implications.

PUBLICATIONS

Morrow, M. E., & Lee, D. (2019). Implementing individualized learning in a legacy learning management system: A feasibility prototype for an online statistics course. *International Journal of Designs for Learning*, 10(1), 131-144.

PRESENTATIONS AT PROFESSIONAL MEETINGS

Morrow, M. (2016). Delivering competency-based, personalized learning through a legacy LMS. Presentation at 2016 Texas Distance Learning Association (TxDLA) Annual Conference, San Antonio, TX.

Morrow, M. (2017). Uncharted Territories: Will Siri replace the instructor in distance education? Presentation at TxDLA 2017 Annual Conference, Galveston, TX, March 30.

Morrow, M. (2017). Using a chatbot to develop reflection in online learners. Poster Presentation at 2017 ISTE Annual Conference, San Antonio, TX, June 28.